UNIVERSITAT POLITÈCNICA DE CATALUNYA

Sensorització portàtil d'edificacions

7 de juliol de 2017

Memòria del projecte que presenta Guillem Serra Padró
sota la direcció del Dr. Eng. Pere Palà
per assolir el grau d'Enginyer en Sistemes TIC.

Vull donar les gràcies a la meva família,
al meu tutor Pere,
al departament de DIPSE,
i a totes les persones que m'han ajudat

Índex

Abstract

This project consists in developing a portable data gathering system in interiors or buildings. The solution consists of temperature, light, and sound sensors each one controlled by a microcontroller. Using another microcontroller, we manage the power supply, sampling time, and data collecting of many sensors. This data is sent to a server that runs on a single board computer through a WiFi network created by the same comupter. The server offers a web interface to configure the system and download the sensor data.

Resum

Aquest projecte consisteix en desenvolupar un sistema portable de recollida de dades en interiors o edificacions. La solució consisteix en sensors de temperatura, llum, i so controlats individualment per un microcontrolador. Utilitzant un altre microcontrolador, gestionem l'alimentació, temps de mostreig, i recollida de mostres de diversos sensors a la vegada. Aquestes dades s'envien a un servidor que s'executa sobre un ordinador mono placa a través d'una xarxa WiFi creada pel mateix ordinador. El servidor ofereix una interfície web per configurar el sistema i permetre la descàrrega de dades.

Part I.

Memòria

1. Introducció

A causa del fenomen de l'Internet of Things, han sorgit al mercat molts productes i empreses dedicades a la recollida de dades utilitzant dispositius autònoms connectats a Internet. Tot i això la majoria d'aquests productes estan enfocats a desenvolupadors o usos professionals, deixant de banda els usuaris no experts en la matèria que busquen solucions que no requereixin coneixements de programació o electrònica per utilitzar-los, ni la contractació d'empreses per realitzar mesures simples. Amb aquest projecte es vol aportar un sistema de recollida de dades enfocat en aquest públic, desenvolupant una solució de baix cost, utilitzable en qualsevol situació, i senzilla de fer servir. En particular, es vol enfocar aquest projecte en les mesures que es realitzin dintre d'edificis o espais coberts.

2. Situació del treball

L'estiu del 2016, l'associació LIMA (Low Impact Mediterranean Architecture) [LIM17], la qual es dedica a l'estudi de l'impacte ambiental en el procés de construcció d'edificis i com disminuir-lo, va demanar a la UPC un sistema per fer mesures de temperatura en un prototip d'edifici dissenyat per ells per validar els resultats del seu estudi. Després d'un primer prototip vaig tenir l'opció de desenvolupar aquesta idea per acabar de dissenyar un sistema més complet.

2.1. Objectiu

L'objectiu d'aquest projecte consisteix en desenvolupar un kit portàtil de sensors, de preu assequible i configurables a través d'una aplicació web. Aquest kit ha de ser prou senzill d'utilitzar perquè el pugui fer servir una persona no experta en la matèria i el seu desplegament ha de consistir tan sols a situar els sensors en el lloc que es desitja i connectar els elements al corrent, fent ús d'una bateria o directament a la xarxa. També es vol aconseguir un consum molt baix per part dels sensors per tal que puguin funcionar dies amb una bateria externa.

A més a més, es vol aconseguir dissenyar una solució que pugui ser extensible fàcilment fent ús de components i software de codi obert, donant l'opció de servir com a base per desenvolupar un sistema més complex o afegir altres components que compleixen amb el protocol que segueixen les diferents parts del sistema.

2.2. Estudi de mercat

Actualment existeixen molts productes en l'àmbit de xarxes de sensors sense fils i plataformes en línia per a la recollida de dades de sensors autònoms i el nombre de productes no para de créixer degut a l'anomenat IoT (Internet of Things). Aquests sistemes es basen en l'ús de *motes*, els quals solen ser un mòdul amb un sensor, un microcontrolador, i un dispositiu de comunicació sense fils per enviar les dades a un dispositiu que s'encarrega de la seva gestió o d'enviar-les a un servidor. Alguns exemples d'aquest sistema de funcionament serien els productes de National Instruments [Ins17] o Advanticsys [Sys17a], els quals ofereixen equips de sensors de temperatura, llum, humitat, etc, però enfocats a un ús industrial o professional amb preus molt més elevats del que es pot considerar acceptable per complir els nostres objectius.

Un altre tipus de funcionament seria el que ofereixen plataformes en línia com Kaa [Kaa17], les quals proporcionen un SDK (Software Development Kit), deixant a l'usuari tota la part de desenvolupament del hardware. També hi han solucions que integren els dos sistemes exposats, és a dir, una plataforma en línia i els dispositius sensoritzadors, però solen ser enfocats a usuaris amb coneixements de programació o electrònica.

En definitiva, podem observar que hi ha una mancança d'un sistema que no requereixi

configuració per part de l'usuari i estigui enfocat a una persona no experta en l'àmbit de dispositius programables o xarxes de comunicació.

2.3. Plantejament de la solució

Per aconseguir els objectius que ens hem proposat, el sistema ha de complir un seguit de requisits:

- Recollida de dades que es puguin identificar de manera única, és a dir, poder identificar els sensors i a on els tenim situats.
- Proporcionar una interfície fàcil d'utilitzar i accessible amb qualsevol dispositiu per controlar i gestionar els diferents sensors.
- Lliurar a l'usuari de realitzar connexions o configurar els sensors més enllà d'assignar un nom i temps de mostreig.
- Tenir una interfície de recollida de dades clara i independent dels dispositius físics.
- Utilització d'elements de baix consum i optimització d'aquests per aconseguir el mínim consum possible.
- Dispositius i elements de mida reduïda per assegurar la portabilitat del sistema

Com ja hem observat en l'estudi de mercat, la majoria de sistemes utilitzen sensors controlats per un microcontrolador i un dispositiu de comunicació sense fils per enviar les dades a un intermediari entre el servidor on es guarden les dades i els sensors. Seguint aquesta idea, el nostre sistema també estarà compost de diversos sensors acoblats a un microcontrolador (figura 2.1) ja que necessitem adaptar els diferents protocols de comunicació que s'utilitzen a un protocol únic que puguem controlar i entendre amb el nostre sistema.

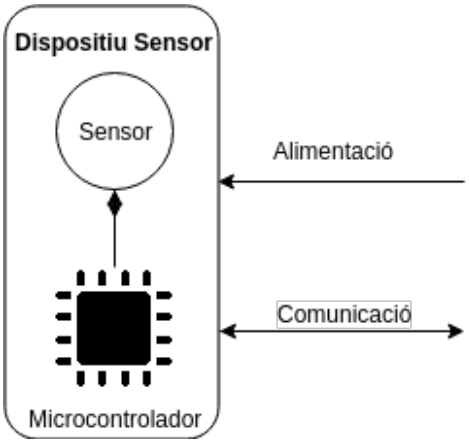

Figura 2.1.: Esquema del dispositiu sensor.

A diferència dels altres sistemes, el nostre delegarà la transmissió de dades a un dispositiu comú entre uns quants sensors (figura 2.2) el qual també controlarà els temps de mostreig i el consum dels sensors. Això ens permetrà simplificar el sistema, delegant tota la gestió a un

dispositiu i creant un sol punt d'alimentació per diversos dispositius, permetent abaratir el cost del sistema en general. Aquest dispositiu es comunicarà amb el servidor per rebre ordres i transmetre les dades.

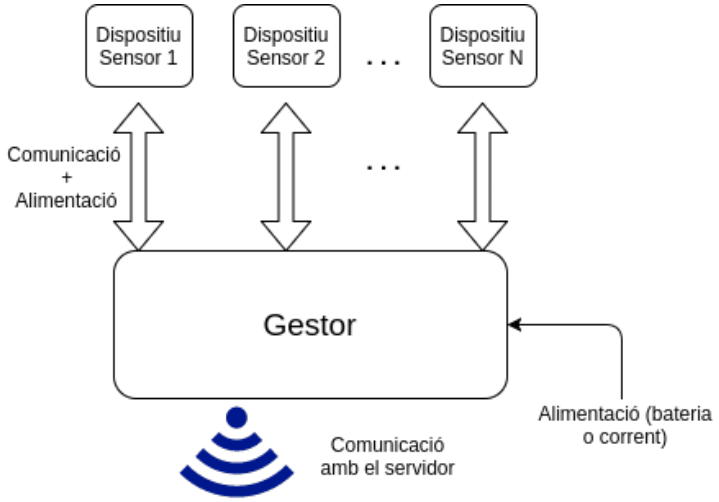

Figura 2.2.: Esquema del dispositiu gestor.

Ja que volem un sistema completament portàtil i que no requereixi configuració, el servidor no estarà a Internet, sinó que estarà en un dispositiu que s'encarregarà d'oferir la interfície web per controlar els sensors i recollir les dades pel seu posterior processat (figura 2.3). Això permetrà que aquest sistema funcioni sense connexió a internet, però també ofereixi la possibilitat d'accedir al control dels sensors a través d'Internet fent ús de la web.

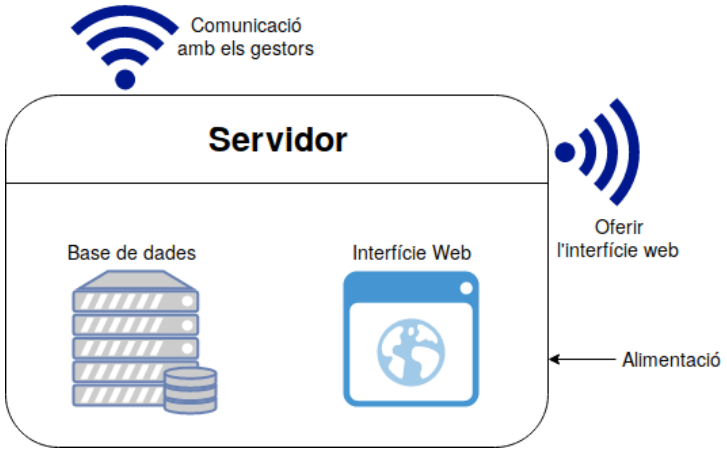

Figura 2.3.: Esquema del dispositiu servidor.

En definitiva, un funcionament típic del sistema seria el següent (figura 2.4):

1) S'endolla el servidor a la corrent.

2) S'alimenta el dispositiu gestor amb una bateria o directament a una presa.

3) Es connecten els sensors al gestor.

4) Es configura el temps de mostreig, localització dels sensors, etc, a través de la web.

5) Es comença la recollida de dades.

6) Consultem les dades recollides a través de la web.

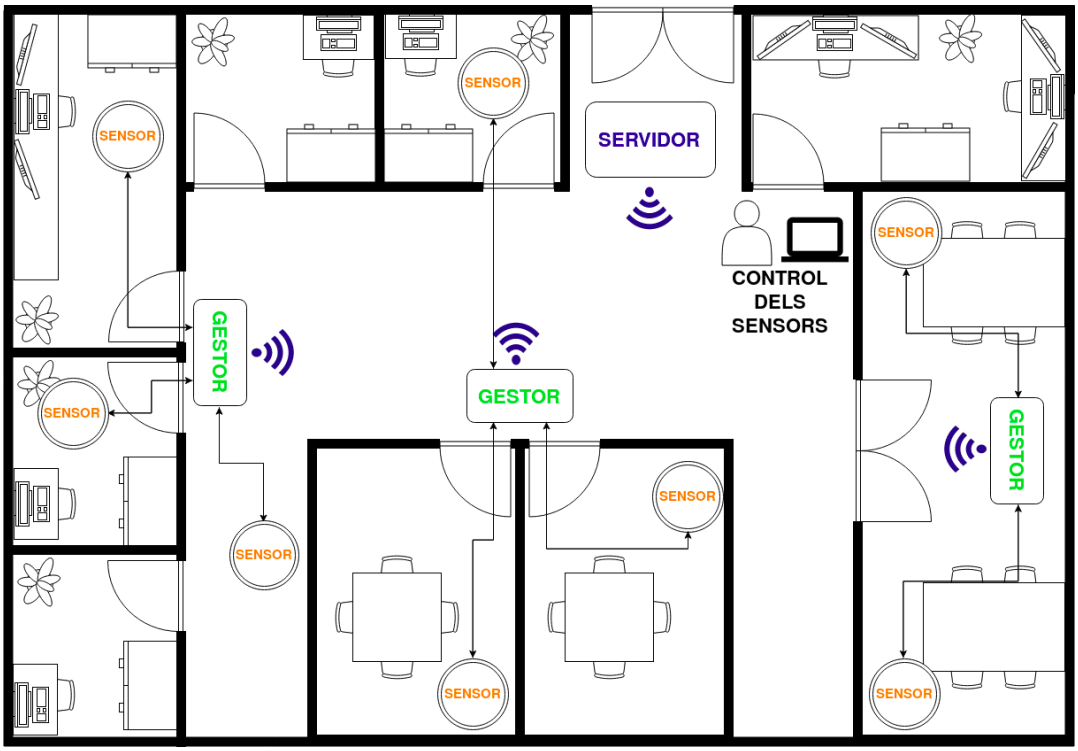

Figura 2.4.: Exemple de funcionament del sistema complet.

3. Estudi previ

Com ja hem explicat en l'estudi de mercat, el sorgiment de l'IoT i la constant millora de la potència de processat en dispositius cada vegada més petits i amb menys consum, ha tingut com a conseqüència la creació d'un ventall de productes enfocats a la recollida de dades i a la comunicació entre dispositius autònoms. A causa d'això, hem hagut d'explorar diverses opcions a l'hora de decidir quins d'aquests productes s'adapten millor a les necessitats del projecte. En general, es busquen components amb les següents característiques:

- Consum i mida reduïts.
- Elements que hagin estat provats i gaudeixin d'una bona documentació i una bona base de software i llibreries.
- Software de codi obert i sistemes oberts de hardware, per poder integrar correctament els dispositius al sistema.
- Productes de baix cost.

3.1. Sensors

En aquest àmbit és en el que tenim més possibilitats per triar, per la gran varietat de sensors de tots tipus i per moltes necessitats diferents. En el nostre projecte el que busquem dels sensors és, sobretot, que utilitzin un protocol de comunicació compatible amb un microcontrolador i que no requereixin una gran configuració. Tenint present que un dels nostres objectius és crear un kit assequible, hem de limitar la cerca a sensors de l'àmbit no industrial però amb una qualitat suficient per assegurar la qualitat de les dades recollides.

Hi ha moltes magnituds que podríem mesurar, però en aquest projecte n'hem triat 3: temperatura, llum, i so. Amb això es vol demostrar la necessitat de tenir un microcontrolador per cada sensor i l'avantatge que suposa delegar la lògica de presa de dades al microcontrolador, donant l'opció d'integrar sensors de molts tipus al sistema.

3.1.1. Temperatura

Hi ha molts projectes que es basen en lectura de temperatura i en la majoria d'ells s'utilitza un d'aquests dos sensors: DHT22 [Max17](figura 3.1) o DS18B20 [Int17b](figura 3.2). Els dos són sensors de baix cost i amb una interfície fàcil de llegir amb un microcontrolador.

A la taula 3.1 podem observar les diferències entre els dos sensors. Tant en precisió, rang de mesura, característiques elèctriques, i interfície de comunicació, els dos sensors són adequats pel nostre sistema. La diferència cau sobre el preu i la mida, en què el DS18B20 sembla que es podria integrar millor amb el microcontrolador. A més a més, nosaltres només volem mesurar

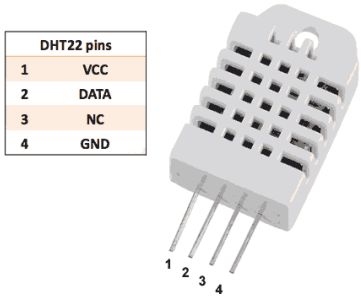

Figura 3.1.: Sensor DHT22 amb encapsulat protector.

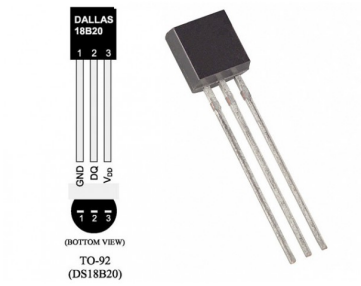

Figura 3.2.: Sensor DS18B20.

la temperatura, ja que fer servir el DHT22 per mesurar la humitat i temperatura a la vegada complicaria la implementació d'un protocol transparent al sensor. Al final ens hem decantat per utilitzar el sensor DS18B20.

3.1.2. Llum

El sensor de llum que necessitem nosaltres és per calcular la llum ambient. Aquests sensors solen funcionar fent ús de fotodíodes, els quals transformen la quantitat de llum a corrent elèctric. En el nostre cas hem optat per no fer servir els productes que ofereixen sensor de llum sobre plaques de circuit ja muntades, sinó que utilitzarem directament el sensor de llum i llegirem les mostres seguint el protocol que faci servir. Després de buscar entre sensors de llum ambient, els dos candidats finals són: APDS-9301 [Tec10], figura 3.3, i TSL2591 [AG15], figura 3.4. A la taula 3.2 podem observar que són molt semblants i segurament podem fer servir els dos sensors, però el TSL2591 és un sensor que s'utilitza en un mòdul bastant conegut [Ada17], per tant és més probable que trobem llibreries i documentació de com fer-lo servir, per tant utilitzarem el sensor TSL2591 per mesurar la llum ambient.

3.1.3. So

Per capturar el so ambient i obtenir una lectura en dB, hem decidit utilitzar una breakout board [Spa17b] (figura 3.5), amb un micròfon electret [Ele10], el qual capta l'amplitud de so fent servir dues plaques conductives, una vibrant i l'altra fixa, i un amplificador [Ins08]. Aquesta placa ens permet obtenir lectures a partir d'una sortida analògica, la qual s'ha de

	DHT22	DS18B20
Preu	6-9€	3€
Rang	-40°C-80°C	-55°C->;;;125°C
Precisió	+-0.5°C	+-0.0625°C
Mida	15.1mm*25.1mm*7.7mm	5mm*4mm*5mm
Connexions	3 pins	2/3 pins
Voltatge DC	3.3V-6V	3V-5V
Consum	1.5mA	1.5mA
Sortida	Maxim 1 wire	Dallas 1 wire
Temps lectura	>2s	750ms

Taula 3.1.: Taula comparativa entre sensors de temperatura.

	APDS-9301	TSL2591
Preu	1.25€	1.62€
Rang	0-65000 lux	0-88000 lux
Sensitivitat	0.1 lux	0.000188 lux
Mida	2.6mm*2.2mm*0.55mm	2mm*2.4mm*0.65mm
Connexions	6 pins	6 pins
Voltatge DC	2.7V-3.6V	2.7V-3.8V
Consum	0.24mA	0.27mA
Sortida	I2C	I2C

Taula 3.2.: Taula compartiva entre sensors de llum.

llegir a través d'un ADC (Analog to Digital Converter). Aquest dispositiu es pot utilitzar de vàries formes: detectar events de soroll, mesurar amplitud en una certa freqüència, etc. L'ús que se'n faci depèn de com es llegeixin els seus valors, per tant no ens ha de suposar un problema obtenir una lectura en dB. A la taula 3.3 podem observar que té les característiques adequades pel nostre projecte (mida, preu, baix consum) i per tant hem decidit no buscar alternatives.

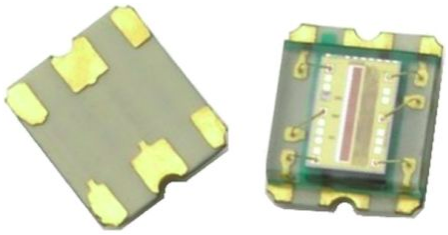

Figura 3.3.: Sensor APDS-9301.

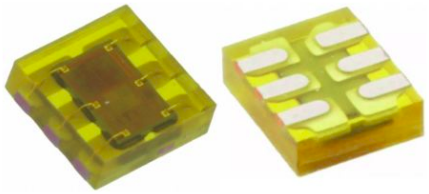

Figura 3.4.: Sensor TSL2591.

	Micròfon electret
Preu	7€
Rang	100Hz-10kHz
Sensitivitat	-44 dB
Mida	21mm*11mm*7mm
Connexions	3 pins
Voltatge DC	2.7V-5.5V
Consum	<1mA
Sortida	Analògica

Taula 3.3.: Taula característiques micròfon electret.

3.2. Microcontroladors de baix consum

Els microcontroladors de baix consum són unitats de processat en un circuit integrat. Solen incloure una CPU (Central Processing Unit), memòria volàtil i no volàtil, sortides i entrades digitals, i són programables. La seva mida reduïda i el seu baix consum permet integrar-los en projectes molt variats realitzant tasques molt diferents. En el nostre projecte busquem microcontroladors amb les següents característiques:

- Facilitat per programar-lo i un bon suport de llibreries.
- Suport per als protocols de comunicació més comuns: I2C, SPI, Serial.
- Suficients pins d'entrada i sortida.
- Convertidor analògic-digital i comptadors.

Tal com hem exposat en l'apartat 2.3, utilitzarem dos microcontroladors: un per controlar cada sensor individualment, i un que farà la funció de preparar les dades per enviar-les al servidor.

3.2.1. Microcontrolador del sensor

Per llegir les dades dels sensors necessitem un microcontrolador que suporti els protocols més comuns en els sensors digitals (I2C, SPI) i que també tingui un ADC per poder utilitzar sensors analògics, com per exemple el micròfon. Existeixen molts microcontroladors que compleixen els nostres requisits, però ens decantarem per la família de microcontroladors AVR, més específicament per els ATtiny [Atm17], els quals són especialment pensats per utilitzar en casos

Figura 3.5.: Placa amb micròfon electret.

que es necessita un microcontrolador petit i de baix consum. A més a més, al tenir una estructura similar als altres microcontroladors de la família AVR, són compatibles i programables de manera similar als microcontroladors que hem utilitzat durant la carrera.

Un cop tenim clara la família de microcontroladors que volem utilitzar n'hem de triar un dintre de les moltes possibilitats. Hem decidit utilitzar una placa de circuit imprès basada en el ATTiny85 [Atm13]: Digispark USB Development Board [Dig17] (figura 3.6). Aquesta placa compleix amb el requisit de baix cost (entre 3 i 7 euros) i ens ofereix avantatges molt importants:

- S'insereix directament a un port USB per ser programat.
- Suport de múltiples llibreries d'Arduino adaptades a ATTiny85.
- 6 pins d'entrada/sortida.
- Esquemàtics de la placa completament accessibles.
- Suport per comunicació I2C i SPI.
- Pins de ADC.

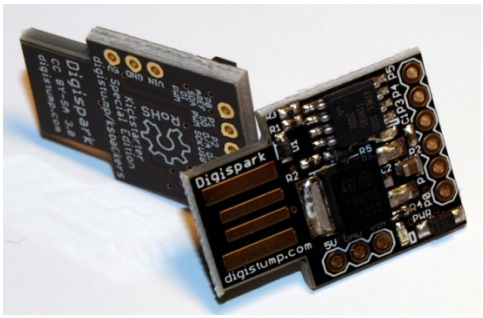

Figura 3.6.: Placa de circuit imprès Digispark.

La programació a través d'USB ens facilita molt el procés de desenvolupament, ja que és una manera ràpida i senzilla de programar el dispositiu. A més a més, el fet que suporti moltes llibreries ja existents ofereix un gran ventall de possibilitats a l'hora de decidir quines funcionalitats realitzarà el microcontrolador. A part de les opcions que ens ofereix la placa, l'ATTiny85 disposa de:

- 8 KBytes de memòria flash programable, 512 Bytes de memòria EEPROM (Electrically Erasable Programmable Read-Only Memory) programable, i 512 Bytes de memòria SRAM (Static Random-Access Memory).
- USI (Universal Serial Interfece), el qual ens permet fer ús dels protocols Serial, I2C, i SPI.
- Clock intern.
- ADC de 10 bits.
- Alimentació entre 2.7V i 5.5V.

Amb aquestes característiques podem assegurar que la placa Digispark serà capaç de funcionar amb la gran majoria de sensors en el rang de preu que ens movem.

3.2.2. Microcontrolador del gestor

El microcontrolador gestor s'encarregarà de:

- Demanar les dades dels sensors als microcontroladors ATTiny85.
- Controlar el temps de mostreig dels sensors.
- Gestionar l'alimentació dels sensors.
- Comunicar-se amb el servidor i actuar en conseqüència.

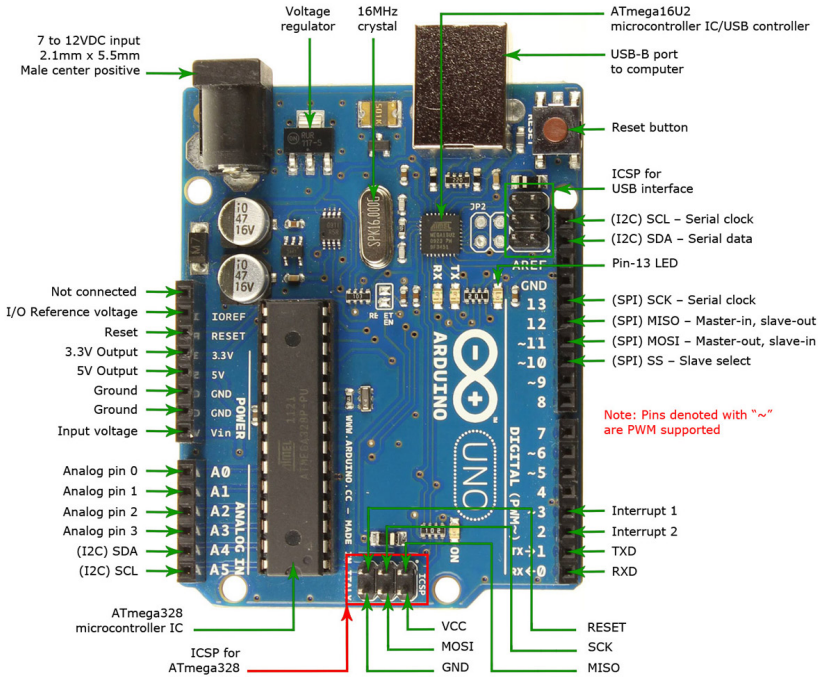

Figura 3.7.: Placa de circuit imprès Arduino UNO.

Per complir aquestes funcions no necessitem gaire potència de processament, si que és necessari tenir més pins d'entrada/sortida que en els sensors. Igual que hem triat una placa de circuit imprès pels sensors degut a les grans facilitats que ens proporciona a l'hora de desenvolupar i programar el sistema, en aquest cas escollirem la placa Arduino UNO [Ard17a] (figura 3.7). Aquesta placa té les següents característiques rellevants pel nostre projecte:

- Programació a través d'USB.
- Moltes llibreries públiques de sensors, protocols de comunicació, mòduls de comunicació sense fils.
- És un producte molt estès, fent que hi hagi molta informació i documentació sobre com realitzar funcions molt diverses.
- 14 pins d'entrada/sortida.
- Permet alimentació a través d'USB, alimentador de 2.1mm x 5mm, o a través d'un pin.
- Pot proporcionar fins a 150mA de corrent i 3.3V o 5V.
- 32 KBytes de memòria flash programable.
- Preu entre 5 i 20 €.

Aquesta placa utilitza el microcontrolador ATMega328p [Atm15], el qual també és de la família AVR, per tant, com ja hem dit abans, és compatible amb l'ATTiny85, fent que puguem tenir un entorn de desenvolupament de codi molt més senzill. També hem de tenir en compte que com hem explicat en l'apartat 2.3, aquesta placa haurà d'alimentar als sensors, per tant el fet que pugui subministrar fins a 150mA pels pins d'alimentació i fins a 40mA pels pins digitals fa que sigui un candidat adequat pel projecte. A més a més, el fet de tenir 2 anys d'experiència treballant amb aquesta placa en diferents projectes fa que puguem tenir un coneixement de què es pot fer i com s'ha de fer.

3.3. Protocols de comunicació

Decidir com es comunicaran els diferents elements del sistema és una part vital, ja que marcarà i condicionarà moltes de les decisions que prenguem a l'hora de programar els dispositius i de les funcionalitats que podrem donar al sistema. En el nostre cas podem diferenciar dos tipus de comunicacions: entre el gestor i els sensors, i entre el gestor i el servidor. En el primer cas utilitzarem una comunicació amb fils, a causa de la necessitat d'alimentar els sensors des del gestor. En l'altra cas utilitzarem una comunicació sense fils per tal de facilitar el desplegament de sensors en els llocs que es desitgi de l'edifici.

3.3.1. Comunicació entre sensors i gestor

En els apartats anteriors ja hem anomenat els protocols de comunicació més comuns dels que disposen els microcontroladors: I2C, SPI, i TTL Serial. Aquest protocol són els que hem estudiat més durant la carrera i tots ells permeten comunicació entre múltiples dispositius, però ho fan de maneres diferents.

Començarem explicant com funciona el protocol I2C (Inter-Integrated Circuit). És un protocol síncron, half-duplex (només es poden enviar dades en un sentit a la vegada), amb un esquema de funcionament mestre-esclau. Només necessitem 3 cables per comunicar tots els dispositius: SCL (clock), SDA (dades), i alimentació. El procés per enviar dades és el següent:

1) El dispositiu mestre deixa el pin SCL en estat alt i posa el pin SDA en estat baix. Això indica el començament d'una transmissió.

2) S'envien 7 bits indicant l'adreça del dispositiu amb qui es vol parlar i 1 bit indicant si és una operació d'escriptura o de lectura.

3) S'envien les dades. A cada paquet de dades enviat, el dispositiu esclau ha de respondre posant en estat baix el pin SDA, si això no succeeix, es considera que no s'han rebut les dades i es pot parar la transmissió.

4) S'indica fi de transmissió amb un canvi d'estat en el pin SCL de baix a alt, i després el mateix amb el pin SDA.

Com el nom indica, és un protocol pensat per comunicació entre circuits a distàncies molt curtes, i en el cas particular del ATMega328p, s'especifica que per utilitzar el protocol I2C hi ha d'haver una capacitància inferior a 400pF. A causa del fet que la capacitància augmenta com més dispositius estiguin connectats en els tres cables comuns, i com més distància hi hagi entre els dispositius, de seguida arribaríem al límit. Per exemple un cable 22 AWG pot tenir una capacitància d'uns 100 pF/m [Bel12]. Com podem veure el protocol I2C no és adequat per la comunicació entre els sensors i el gestor però sí que l'haurem d'utilitzar per llegir dades de sensors, per tant hem de saber com funciona igualment.

El següent protocol és el protocol SPI (Serial Peripherial Interface). És un protocol de comunicació sèrie síncron i full-duplex (permet l'enviament en els dos sentits a la vegada), amb un esquema de funcionament mestre-esclau [Gam17]. Requereix 4 pins per funcionar més 1 pin extra per cada esclau que es vol afegir (figura 3.8):

- *SCLK* (Serial Clock): és el clock que s'utilitza a l'hora de realitzar les comunicacions. El controla el dispositiu mestre i es pot compartir entre tots els esclaus.

- *SS* (Slave Select): s'utilitza per seleccionar quin dispositiu esclau se li vol enviar o rebre dades. No es pot compartir entre els esclaus.

- *MOSI* (Master Output Slave Input): és el canal d'enviament de dades del mestre cap a l'esclau.

- *MISO* (Master Input Slave Output): és el canal d'enviament de dades de l'esclau cap al mestre.

El problema amb aquest protocol és que està pensat per comunicacions en distàncies curtes. Com que utilitza la línia SCLK per sincronitzar la transmissió de dades, a l'afegir distància entre els dos dispositius també estàs afegint un retràs en aquest senyal, fins a arribar al punt que això provoca la des sincronització dels dispositius. Aquest problema es pot solucionar

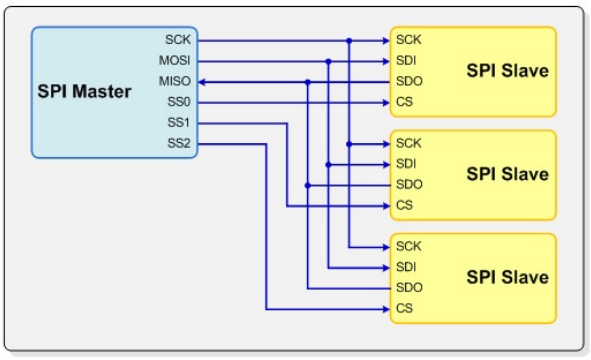

Figura 3.8.: Connexions del protocol SPI amb múltiples esclaus [Cor17].

afegint repetidors i circuits extra, tal com s'explica en aquest document de Thomas Kugelstadt [Kug11], però això complicaria massa el nostre projecte. Degut a això descartem l'ús de SPI.

Per acabar, l'última opció és el protocol de comunicació TTL (Transistor-Transistor Logic) serial [Spa17a]. A diferència dels altres dos protocols, es tracta d'un protocol asíncron, és a dir que no requereix un senyal de clock comú entre els dispositius. En total es necessiten 3 cables: RX (rebre dades), TX (transmetre dades), i GND. Tot i no tenir un clock comú, si que és necessari que els dispositius que es volen comunicar es posin d'acord en una sèrie de paràmetres:

- *Bits de dades*: es pot triar entre 5 a 9 bits, però el més comú és enviar 1 byte (8 bits).
- *Bits de sincronització*: s'envien amb cada paquet de dades, i serveixen per marcar el principi i final del paquet.
- *Bits de paritat*: comprovar la paritat del paquet de dades serveix com a forma senzilla de control d'errors. El que es fa és mirar el nombre d'1 en el paquet, i s'envia 1 bit indicant si aquest nombre és parell o senar. Aquest control és opcional.
- *Baud rate*: aquest paràmetre marca la velocitat a la qual s'envien els paquets de dades. En el microcontrolador ATMega328p es poden configurar moltes velocitats, però les més típiques solen ser 9600 bits/s o 115200 bits/s.

L'element encarregat d'organitzar els paquets de dades i enviar tots els bits de control és l'UART (Universal Asynchronous Receiver/Transmitter). Pot ser un circuit integrat a part o estar inclòs dintre el microcontrolador, com és el cas de l'ATMega328 que disposa d'una UART. També es pot implementar el comportament d'aquest dispositiu directament en el microcontrolador per crear un Software Serial [Ard17c], que bàsicament permet utilitzar qualssevol dos pins digitals com a RX i TX.

El fet que sigui una comunicació sèrie TTL implica que els nivells de voltatges durant la transmissió estaran en el rang dels valors d'alimentació del microcontrolador, que en el nostre cas és de 5V. A la figura 3.9 es pot veure un exemple de transmissió d'un paquet de dades de 8 bits. Ja que es tracta de transmissió de dades a baixa velocitat, no tindrem problemes

amb el rang de distància en que estem treballant distància (1 a 10 metres). Si es detectessin problemes, es pot utilitzar un cable amb menys capacitància o reduir encara més la velocitat de transmissió.

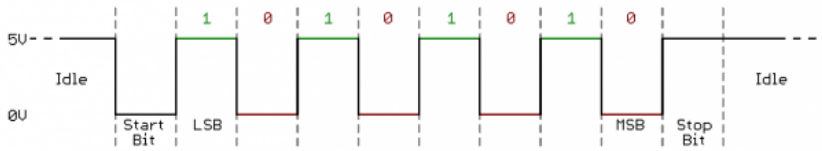

Figura 3.9.: Transmissió de dades utilitzant protocol sèrie TTL [Spa17a].

És cert que també podríem aplicar la mateixa lògica amb el protocol SPI, és a dir utilitzar millors cables per assegurar la bona comunicació, però ja que amb els dos protocols podem connectar el mateix nombre de dispositius, i que durant la carrera hem fet servir molt més el protocol TTL Serial que no pas SPI, ens decantarem per utilitzar el primer.

3.3.2. Comunicació entre gestors i servidor

Per realitzar la comunicació entre els gestors i el servidor hem decidit fer-ho sense fils. Considerarem 3 opcions: Bluetooth, RF, i WiFi. Per seleccionar el candidat observarem les següents característiques:

- Baix consum i modes de parada total o parcial.
- Abast de les comunicacions.
- Integració senzilla al sistema.

Per fer la comparació entre els tres hem seleccionat tres mòduls concrets, els quals estan en el rang de preus acceptable pel projecte i són bastant comuns: NRF24L01 [Sem08] (figura 3.10), ESP8266 [Tea15] (figura 3.11), i HC-05 [Stu10] (figura 3.12).

	HC-05	NRF24L01	ESP8266
Preu	2-6€	3-8€	3-6€
Freqüència	2.4GHz	2.4GHz	2.4GHz
Abast	10m	250m	350m
Voltatge DC	1.8V-3.6V	1.9V-3.6V	3V-3.6V
Consum (actiu)	30mA	12mA	130mA
Consum (mínim)	2mA	900nA	20uA
Mida	12.7mm*27mm	15mm*29mm	14mm*30mm
Interfície	TTL Serial	SPI	TTL Serial

Taula 3.4.: Taula comparativa entre mòduls de comunicació sense fils.

Observant la taula 3.4, podem concloure que les opcions més bones són el NRF24L01 o el ESP8266. Els dos tenen un abast més que suficient pels nostres objectius, però el NRF24L01 té un consum molt inferior al ESP8266. Per una banda pot semblar que seria el candidat

perfecte, ja que compleix amb tots els nostres requisits, però hem decidit utilitzar el ESP8266 pel motiu següent: dóna accés al *stack* de comunicació TCP/IP. Això no només significa que tenim l'opció de controlar aquests dispositius des de qualsevol punt amb accés a Internet, sinó que també comporta que ens puguem comunicar directament amb el servidor web sense haver d'adaptar el mètode de comunicació que utilitzi el dispositiu RF.

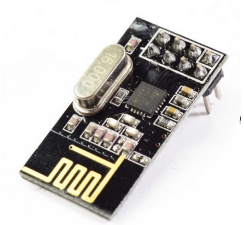

Figura 3.10.: Mòdul NRF24L01.

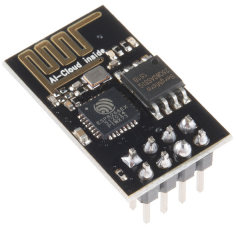

Figura 3.11.: Mòdul ESP8266.

Figura 3.12.: Mòdul HC05.

3.4. Computadors monoplaca

Els ordinadors mono placa, o SBC (Single-Board Computer), són ordinadors construïts sobre una placa de circuit. Solen tenir un microcontrolador, memòria volàtil i no-volàtil, pins d'entrada/sortida, perifèrics, etc. Aporten totes les funcionalitats que pugui realitzar un ordinador comú, però amb una mida molt reduïda. Aquesta última característica és perfecte pel nostre projecte, ja que necessitem un element que actuï com a servidor i gestioni totes les dades dels sensors. Igual que amb els altres components, tenim un ampli ventall d'opcions per triar.

Una de les famílies de dispositius SBC més conegudes són les Raspberry Pi, en concret s'ha decidit utilitzar el model Raspberry Pi 3 Model B [Fou17] (figura 3.13) amb les següents característiques principals:

- 1.2GHz 64-bit quad-core ARMv8 CPU.
- 802.11n Wireless LAN.
- 1GB RAM.
- Connexió Ethernet.
- Ranura Micro SD.
- 4 ports USB.

Gràcies al fet que té connexió WiFi, el podrem utilitzar com a punt d'accés per tal de servir la pàgina web de control dels sensors. El processador és suficientment potent com per gestionar el volum de dades que esperem recollir. Tot i que el processador és de 64 bits, cal tenir en compte que el sistema operatiu oficial que s'ofereix encara és de 32 bits i si bé és cert que existeixen sistemes operatius alternatius que ofereix una arquitectura de 64, nosaltres optarem per utilitzar l'oficial per seguretat i estabilitat.

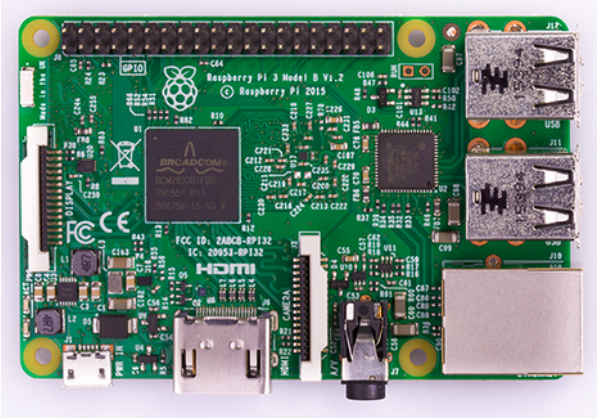

Figura 3.13.: Computador mono placa Raspberry Pi 3 Model B.

3.5. Software

A l'hora d'escollir els llenguatges de programació, base de dades, etc, s'ha valorat que siguin adequats per la seva funció i la familiaritat que es té.

3.5.1. Microcontroladors

Per programar els microcontroladors tenim dues opcions: llenguatge C o entorn de programació Arduino IDE (C++). Si utilitzéssim C, podríem tenir molt més control sobre l'estructura i les funcions del programa, ja que és un llenguatge de baix nivell, a costa de més dificultat a l'hora de programar. Com que no volem realitzar operacions gaire complexes i que no són exigents amb el temps d'execució, no obtindríem gaire benefici en utilitzar C. D'altra banda, si utilitzem l'Arduino IDE, perdrem control sobre què s'està executant exactament en els microcontroladors. El gran avantatge que ofereix l'Arduino IDE és la gran quantitat de llibreries

que existeixen per treballar amb mòduls i shields externs, a més a més de tenir llibreries per la majoria de protocols de comunicació. Un altre aspecte a tenir en compte és que els mateixos fabricants de la placa Digispark, la qual utilitzem per controlar els sensors, ofereixen tot l'entorn de programació de la placa i llibreries adaptades a la placa dintre l'Arduino IDE, per tant sembla que utilitzar l'Arduino IDE és l'opció més adequada.

3.5.2. Bases de dades

Per triar la base de dades hem de tenir en compte l'objectiu d'aquesta, que en el nostre cas és guardar un seguit de dades dels sensors, identificant de manera única cada sensor i sabent a quina hora s'ha agafat la mostra. Últimament s'està estenent l'ús de bases de dades NoSQL, les quals ofereixen més flexibilitat a l'hora d'inserir dades, ja que no requereixen tenir definit un esquema de dades com en les bases de dades SQL. A l'hora de decidir quin dels dos tipus és millor, tot es resumeix en quines necessitats estenen. Si es vol una relació entre dades i realitzar operacions de consulta complexes i assegurar-ne l'atomicitat, normalment sol ser millor utilitzar SQL.

En el nostre cas, volem guardar dades de sensors i alguns paràmetres com temps de mostreig, localització, identificadors de sensors, etc. Ja que els temps de mostreig no són sempre els mateixos tenim dues opcions: guardar cada dada amb una marca de temps o aprofitar la no estructura de les bases de dades NoSQL per tenir estructures de dades diferents per cada sensor. Tal com explica Jay Runkel en aquesta presentació [Run14], a l'utilitzar la base de dades MongoDB[Mon17], podem guardar dades de sensors en paquets d'hores, reduint molt l'espai necessari i el temps de consulta de dades. Per exemple, si guardem les dades amb un esquema com el següent:

```
{
    timestamp: ISODate("2013-10-10T23:06:37.000Z"),
    type: "temperature",
    value: 1000000
},
{
    timestamp: ISODate("2013-10-10T23:06:38.000Z"),
    type: "temperature",
    value: 15000000
}
```

Observem que tenim repetició innecessària dels camps identificadors. En canvi si modifiquem l'esquema per organitzar les dades en paquets de minuts:

```
{
    timestamp_minute: ISODate("2013-10-10T23:06:00.000Z"),
    type: "temperature",
    values: {
        0: 999999,
        ...
        37: 1000000,
        38: 1500000,
```

```
        ...
        59: 2000000
    }
}
```

Evitem repetir els camps identificadors i a l'hora de llegir les dades d'una hora només hem de llegir 60 entrades contra les 3600 que hauríem de llegir que si seguíssim l'esquema de la primera figura. Això també comporta avantatges a l'hora d'inserir dades, ja que es pot fer una inserció d'un bloc sencer i després anar actualitzant els valors a mesura que entren. Aquesta estratègia es pot repetir fins al nivell que vulguem, és a dir crear paquets d'hores, dies, etc.

L'únic inconvenient d'utilitzar MongoDB és que no té suport oficial per sistemes operatius de 32 bits, tot i que es poden trobar paquets d'instal·lació. Aquestes versions adaptades a 32 bits tenen la mida de la base de dades limitada a 2GB. Això pot semblar un gran inconvenient, però més endavant (apartat 4.3.3) veurem que aquesta mida és suficient per a la quantitat de dades que recollim.

3.5.3. Aplicació Web

Per desenvolupar l'aplicació web que s'encarregarà de configurar els gestors i comunicar-se amb la base de dades per inserir o extreure'n mostres, utilitzarem un microframework de Python anomenat Flask [Ron17a]. Les funcionalitats més importants que ens ofereix són:

- Inclou un petit servidor que podem utilitzar si no es requereix molta potència.
- Cada petició HTTP es tradueix en l'execució d'una funció Python, aportant molta flexibilitat a l'hora de programar.
- Permet l'ús del llenguatge Jinja2 [Ron17b], el qual serveix per generar documents HTML dinàmics.
- Accedir de manera senzilla als paràmetres de les peticions HTPP.
- Gaudeix d'una àmplia documentació i és completament open-source.

Gràcies al fet que cada petició es tradueix en una funció de Python (figura 3.14), podem definir de manera molt clara el protocol de comunicació entre els gestors i el servidor. A més a més, podem definir URLs dinàmiques i agafar-ne parts per passar-les com a paràmetres a les funcions, permetent crear APIs de manera molt senzilla. A l'hora de programar, podem seguir el fil d'execució del programa de manera clara i facilita molt la resolució de problemes a través d'una opció per debuggar, la qual mostra la traça completa dels errors a través d'una pàgina web.

3.5.4. Docker

Docker és un programa per crear i gestionar aplicacions dintre contenidors de software. Aquests contenidors permeten tenir accés a tot un entorn de software i eines necessàries per fer funcionar una aplicació des de qualsevol màquina que tingui Docker instal·lat. Això evita tots els problemes que hi puguin haver a l'hora de treballar en diferents màquines, facilitant el procés de desenvolupament.

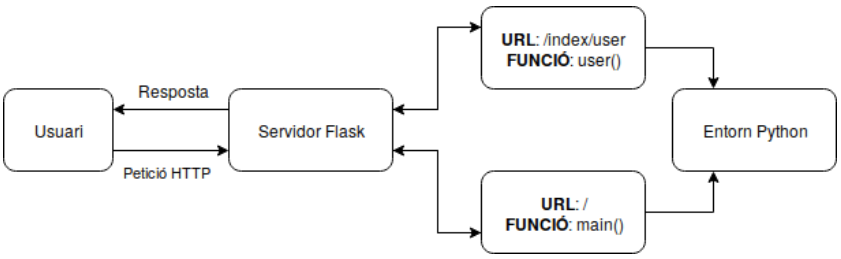

Figura 3.14.: Camí típic que segueix una petició HTTP en Flask.

Els contenidors de software de Docker (figura 3.15) s'executen sobre el sistema operatiu de la màquina *host*, per tant tenen accés als recursos físics com CPU o memòria RAM i múltiples contenidors poden compartir aquests recursos entre ells. Per definir l'entorn que volem dintre el contenidor s'utilitzen fitxers de text anomenats Dockerfiles. La part més important d'aquest fitxer és definir quina imatge base volem. Aquesta imatge conté el sistema de fitxers del contenidor i la podem obtenir a través DockerHub [Doc17c], on podem compartir i trobar imatges per tot tipus d'aplicacions. Docker utilitza un sistema de fitxers COW (Copy-On-Write) [Wik17], per tant si tenim dos contenidors amb la mateixa imatge base, l'espai que ocuparan serà el de la imatge base més les diferències entre els dos. Podem configurar els contenidors de moltes maneres, per més informació es pot visitar [Doc17b].

Normalment necessitem més d'una aplicació, per tant volem tenir múltiples contenidors i sovint també necessitem que es comuniquin entre ells. Per gestionar aquests casos, utilitzem Docker-Compose [Doc17a] perquè ens permet definir quins contenidors volem que s'executin i amb quins paràmetres en un fitxer de text en format YAML, el qual és simplement una manera d'expressar configuracions.

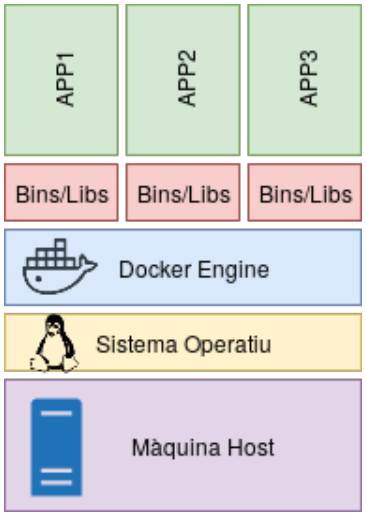

Figura 3.15.: Estructura dels contenidors de software de Docker.

4. Implementació del sistema

El sistema que s'ha implementat s'ha separat en tres parts:

- *Gestors*: són els encarregats de demanar les dades als sensors, alimentar-los, i comunicar-se amb el servidor tant per guardar les dades com per configurar, aturar o engegar el sistema. Estan formats per una placa de circuit imprès Arduino Uno i un mòdul WiFi ESP8266.

- *Sensors*: es comuniquen directament amb els sensors, adaptant el protocol que facin servir a un que puguem gestionar de manera genèrica. Estan formats per la placa de circuit imprès Digispark i un sensor que pot ser de temperatura (DS18B20), de llum (TSL2591), o de so (micròfon electret).

- *Servidor*: el servidor s'encarrega de guardar les dades dels sensors en una base de dades, MongoDB, i oferir una interfície web per gestionar tot el sistema, Flask. A més a més, s'utilitza com a punt d'accés WiFi per tal que els gestors es puguin comunicar amb ell a través de peticions HTTP. Tot això s'implementa sobre un ordinador mono placa Raspberry Pi 3 Model B.

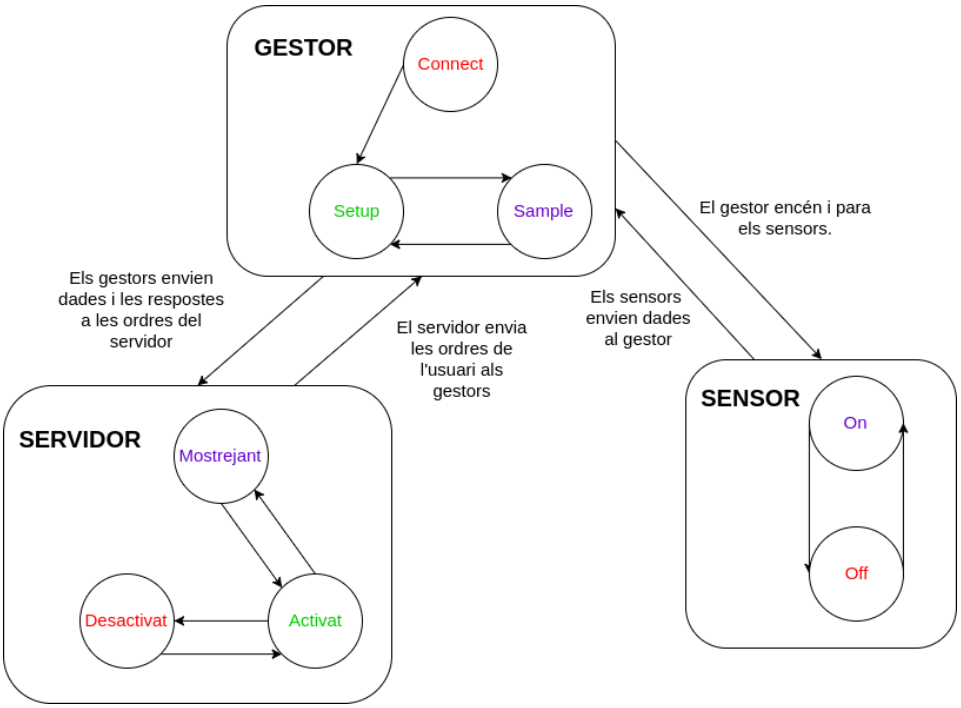

Figura 4.1.: Màquina d'estats sistema en general.

A la figura 4.1 podem observar aquesta estructura. S'ha decidit aquesta implementació per tal de simplificar diversos aspectes. El primer és el fet de tenir un punt central per alimentar un conjunt de sensors, d'aquesta manera es pot controlar més tant el seu consum i podem aprofitar més temps la bateria. Això té com a desavantatge que no puguem assignar un temps de mostreig únic per a cada sensor, però ja que les magnituds que estem mesurant són ambients, és a dir que no pateixen canvis bruscos, no ens suposa una gran pèrdua. Un altre avantatge és assegurar un desplegament de sensors més senzill, ja que en utilitzar directament un cable per connectar els sensors i els gestors, evitem que sorgeixin problemes d'abast o altres que serien difícils de solucionar per una persona no experta en la matèria.

4.1. Gestors

A l'hora de desenvolupar el dispositiu gestor s'ha dissenyat una màquina d'estats (figura 4.2), definint interfícies clares amb les altres parts del sistema per assegurar una bona integració de totes les parts. Com ja hem explicat en l'apartat 3.5.1, s'ha utilitzat l'entorn de desenvolupament Arduino IDE per l'extensa quantitat de llibreries i documentació disponible, a més a més de la facilitat que ofereix a l'hora de programar la placa Arduino UNO. S'ha estructurat el codi en 4 mòduls:

- *gestor.ino*: és el programa principal i a on està definida tota la lògica de la màquina d'estats.
- *utils.ino*: aquest mòdul ens aporta funcions per realitzar certes operacions com per exemple llegir i escriure per un port sèrie en concret, comprovar quins sensors estan connectats, obtenir les mostres dels sensors, etc.
- *wifi.ino*: com el seu nom indica, aquest mòdul conté totes les funcions necessàries per utilitzar el ESP8266. Per exemple, funcions de configuració, construcció de peticions HTTP, activació del mode sleep, etc.
- *pin_definitions.h*: aquest fitxer ens proporciona definicions dels pins que utilitzem. Això aporta claredat i netedat en els altres mòduls i ens permet modificar el funcionament del sistema de manera senzilla.

4.1.1. Màquina d'estats

La part més important del disseny del software és tenir clar el funcionament del sistema sota qualsevol situació. Això es pot aconseguir definint una màquina d'estats abans de començar a escriure codi. En el nostre cas hem definit 3 estats i 5 events (figura 4.2). Començarem explicant quines funcions es realitzen en cada estat:

- Estat *CONNECT*: és l'estat inicial i l'utilitzem per fer la configuració del ESP8266, explicada en l'apartat 4.1.3. Un cop configurat, podem procedir a connectar-nos al punt d'accés creat pel servidor, utilitzant unes credencials fixes. Si no ens podem connectar al punt d'accés per qualsevol motiu, ho tornarem a intentar fins que s'aconsegueixi. Un cop tenim connexió, enviem una petició al servidor per indicar que estem a punt per continuar.

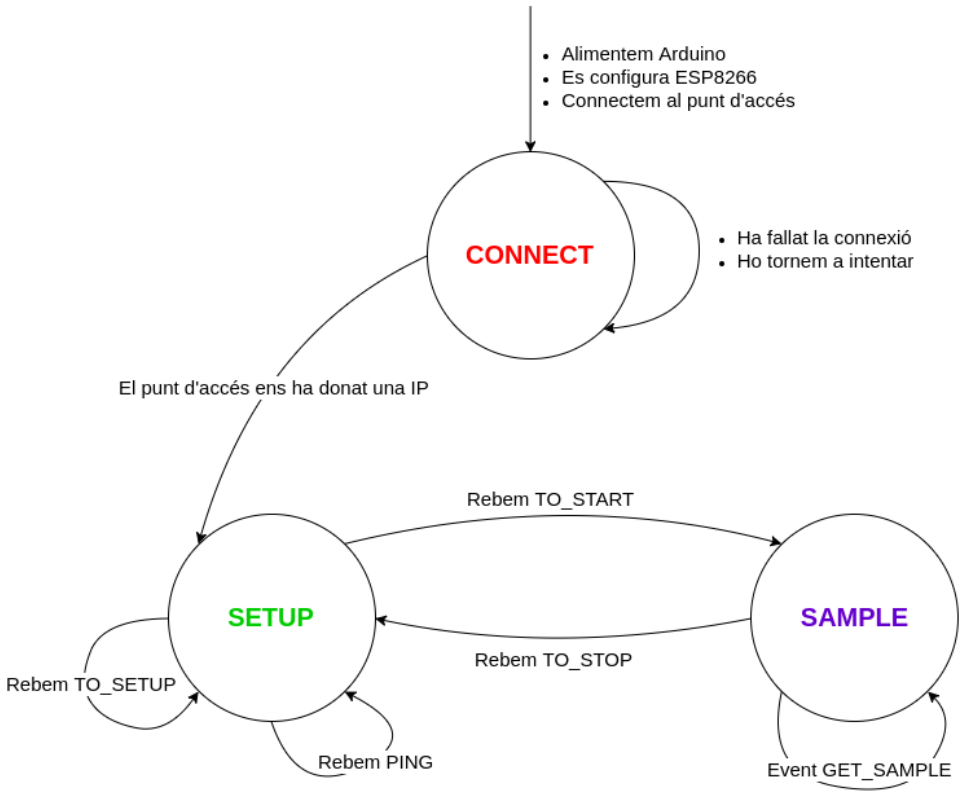

Figura 4.2.: Màquina d'estats del dispositiu gestor.

- Estat *SETUP*: en aquest estat escoltem les ordres que ens pugui enviar el servidor i actuem en conseqüència. Fins que no es rebi l'ordre de començar a agafar mostres, no canviarem d'estat.
- Estat *SAMPLE*: dintre d'aquest estat ens dediquem a recollir les dades dels sensors amb el temps de mostreig que se'ns ha configurat des del servidor. Només sortirem d'aquest estat si rebem l'event de parar just després d'enviar una mostra, enviat des del servidor.

Els events poden tenir dos orígens: servidor o comptador del temps de mostreig. Els provinents del servidor estan explicats en l'apartat 4.1.3. Per generar l'event del comptador, *GET_SAMPLE*, utilitzem la funció *millis()*. Aquesta funció retorna el nombre de mil·lisegons que han passat d'ençà que s'ha començat a executar el programa. Seguint l'estratègia descrita en la guia de James Lewis [Lew17], podem aconseguir comptar el temps de mostreig sense bloquejar en cap moment el programa. També podríem aconseguir-ho fent ús d'interrupcions, però com hem explicat en l'apartat 3.5.1, l'Arduino IDE amaga bastants aspectes de com s'implementen algunes funcions i sol ser més recomanable evitar fer servir interrupcions per no interferir en la possible execució d'altres funcions.

4.1.2. Intefície pels sensors

Per comunicar-nos amb els microcontroladors de cada sensor, utilitzem el protocol sèrie TTL, més concretament utilitzarem la llibreria *SoftwareSerial* [Ard17c], la qual permet comunicaci-

ons en sèrie a partir de 2 pins digitals qualssevol. A més a més d'aquests dos pins també cal que la referència de massa entre els dispositius que es comuniquen sigui la mateixa. Nosaltres disposem de 14 pins digitals, sense comptar els pins analògics que també es podrien fer servir com a digitals però deixarem lliures per tenir la possibilitat d'afegir elements extres de control, però 2 estan ocupats pel *hardware serial*, per tant podem utilitzar 12 pins per comunicar-nos amb els sensors i alimentar-los, és a dir, 3 pins per cada sensor més 1 pin de massa comú. En definitiva, podem tenir 4 sensors connectats al mateix gestor. A l'hora de llegir el que ens enviïn els sensors utilitzarem sempre el mateix format: `GOTipus$dada$`. Gràcies a això no haurem d'indicar si volem demanar una dada o simplement saber de quin sensor es tracta. A més a més no haurem de preocupar-nos de si la mostra del sensor és *float*, enter, positiu, negatiu, etc, perquè nosaltres llegirem un *String* entre les dues marques de delimitació.

Per gestionar els 4 ports sèrie que tenim oberts, crearem una variable la qual serà una estructura on definirem les següents propietats de cada port: apuntador a un objecte *SoftwareSerial* per poder utilitzar les funcions necessàries, pin de RX, pin de TX, pin per alimentar el sensor, variable per indicar si el sensor està connectat, i un *String* on guardarem la mostra de cada sensor. En aquest cas podem observar l'avantatge que suposa l'ús del fitxer *pin_definitions.h* per definir de manera entenedora els pins que es fan servir.

```
// Estructura de dades per tractar els sensors
struct portStruct
{
    SoftwareSerial* portPointer[4] = {&portOne, &portTwo, &portThree, &portFour};
    volatile bool activePort[4] = {false, false, false, false};
    uint8_t portPower[4] = {POWER_ONE, POWER_TWO, POWER_THREE, POWER_FOUR};
    uint8_t portRX[4] = {RX_ONE, RX_TWO, RX_THREE, RX_FOUR};
    uint8_t portTX[4] = {TX_ONE, TX_TWO, TX_THREE, TX_FOUR};
    String mostra[4] = {"","","",""};
} serialPorts;
```

Abans de poder-nos comunicar amb cada dispositiu sensor, hem de saber quins estan actius. Per aconseguir-ho llegirem el pin RX que correspongui, ja que en el protocol sèrie aquest pin sempre està en estat alt quan hi ha un dispositiu connectat. Per assegurar que no detectem un fals positiu, situarem resistències de pull-down en aquests pins. Utilitzarem sempre la mateixa funció per detectar presència dels sensors: *parse_sensors()*. Aquesta funció posarà en estat alt tots els pins d'alimentació i procedirà a llegir els pins de RX. Si detectem estat alt, assignarem *true* a la variable *activePort* que toqui i demanarem que ens enviïn una mostra de dades. D'aquesta mostra extraiem el tipus de sensor per enviar-ho més endavant al servidor. Un cop realitzades totes les lectures tornarem a posar en estat baix tots els pins d'alimentació. Per estalviar energia, quan estiguem mostrejant apagarem el sensor entre lectura i lectura. Com hem dit abans, el pin RX està sempre en estat alt, per tant, si tenim un sensor connectat, hem d'assegurar que des del gestor no s'estan enviant 5V des del pin TX, ja que això provoca que el microcontrolador del sensor no es pari del tot i quedi en un estat inestable que pot provocar danys en el dispositiu. En resum, el procés d'alimentació i detecció de sensors es pot veure en la figura 4.3.

Per llegir les mostres dels sensors seguirem el següent procediment:

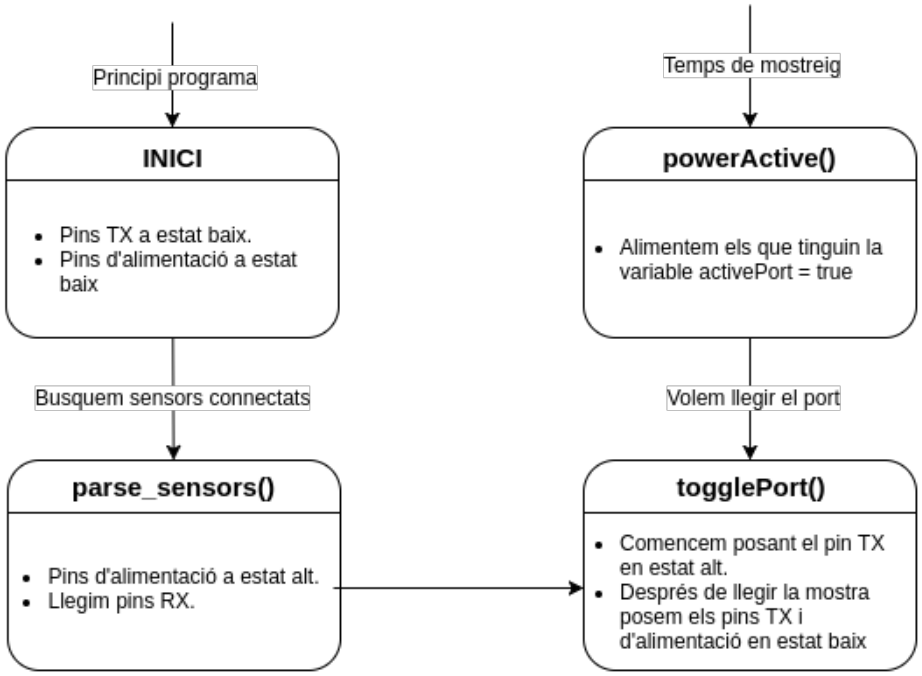

Figura 4.3.: Alimentació i detecció de sensors.

1) Alimentar els sensors actius, és a dir els que s'han configurat, i esperar 100ms per tal que facin les configuracions que calgui.

2) Posem en estat alt el pin TX d'un sensor.

3) Comencem el protocol sèrie amb l'objecte *SoftwareSerial* que toqui.

4) Enviem un missatge al sensor indicant que volem llegir dades.

5) Per extreure les dades dels paquets `GOTipus$dada$` utilitzem la funció *Serial.find()* per detectar el caràcter $ i la funció *Serial.readStringUntil()* per llegir les dades fins el següent caràcter $. Aquestes funcions pertanyen a la classe *Serial* [Ard17b], però les podem utilitzar, ja que la classe *SoftwareSerial* deriva de la *Serial*. Per evitar bloquejar el programa, donem 7 segons per completar la comunicació, si s'excedeix aquest temps, indicarem que hi ha hagut un error retornant una mostra amb valor `ERROR`.

6) Parem la comunicació sèrie i posem en estat baix els pins d'alimentació i de TX.

Un cop hem realitzat tots aquests passos, tenim les dades del sensor guardades en un variable a punt per enviar al servidor.

4.1.3. Interfície pel servidor

Per comunicar-nos amb el servidor utilitzem el mòdul WiFi ESP8266, el qual es comunica amb l'Arduino a través del *hardware serial*. Per enviar ordres a aquest mòdul s'utilitzen comandes AT, les quals són cadenes de caràcters curtes. Les comandes que accepta aquest mòdul es poden consultar en el manual del fabricant [Sys17b]. Per llegir les respostes a les comandes

utilitzarem les mateixes funcions que per llegir dades, *Serial.find()* i *Serial.readStringUntil()*, modificant el temps de *timeout* amb la funció *Serial.setTimeout()*.

Abans de poder parlar amb el servidor hem de configurar el mòdul WiFi de manera que es connecti al punt d'accés i estigui en el mode de funcionament correcte, ja que aquest mòdul proporciona moltes funcionalitats i s'ha de configurar pel nostre cas particular. La configuració que carregarem és la següent:

- *AT+CIOBAUD=115200*: aquesta comanda només l'hem d'enviar un cop, ja que es guarda en memòria. Serveix per definir quina serà la velocitat que anirà el *serial*.
- *AT+CWMODE=1*: definim el mode de funcionament, en aquest cas posem el ESP8266 en *station mode*, que és el mode per connectar-se com a client a una xarxa WiFi.
- *AT+CIPMUX=1*: definim que s'acceptin múltiples connexions. Això ens permetrà tant enviar peticions HTTP com rebre'n.
- *AT+CIPSERVER=1,80*: en conjunt amb la comanda anterior, indiquem que escoltarem les peticions HTTP al port 80.
- *AT+CWJAP=ssid,passwd*: un cop està tot configurat, ens podem connectar al punt d'accés indicant SSID i contrasenya. Si al cap de 10 segons no hem obtingut una IP, tornarem a enviar la comanda fins que ens hi connectem correctament.

Ara que ja estem dins la xarxa del servidor, podem enviar i rebre peticions HTTP, en concret enviarem només peticions GET i rebrem només peticions POST. Per construir les peticions des de l'Arduino, s'han de seguir els següents passos:

1) *AT+CIPSTART=0,TCP,servidor,80*: aquesta ordre indica que volem començar una petició pel canal 0 del ESP8266, al servidor *servidor* (pot ser un nom o directament IP), i pel port 80. Si el ESP8266 respon OK, podem continuar.

2) *AT+CIPSEND=0,llargada*: amb aquesta ordre indiquem la llargada en caràcters que tindrà la petició que volem realitzar. Un cop el ESP8266 ens respon amb el caràcter `>;;;;`, podem enviar la petició, que tindrà el següent format, on utilitzem la URI (Uniform Resource Identifier) per enviar dades:

```
"GET " + uri + " HTTP/1.1\r\n" +
"Host: " + servidor + "\r\n" +
"Connection: close\r\n\r\n";
```

3) *AT+CIPCLOSE=5*: ja que no ens interessa la resposta del servidor, tanquem totes les connexions.

Per llegir les peticions POST que ens envia el servidor, estarem buscant sempre la cadena de caràcters *action*, amb un *timeout* variable que modificarem en funció de si ens interessa escoltar les ordres del servidor o no. Si trobem aquests caràcters, sabrem que hem rebut alguna ordre de les següents:

- *setup*: causa l'event *TO_SETUP*. Si estem en l'estat de *SETUP*, llegirem els paràmetres de configuració amb la funció *Serial.readStringUntil()*, utilitzant el caràcter de delimitació `&`. Les variables que es poden configurar són: temps de mostreig i identificador del gestor. Respondrem al servidor amb una petició GET a l'adreça */gestors/state/synced* per indicar que s'han actualitzat els valors correctament. Assignem 5000 ms com a *timeout* per escoltar les següents ordres del servidor.

- *ping*: causa l'event *PING*. Si no estem en l'estat de mostreig, enviarem al servidor els valors de les variables de configuració que tenim a l'adreça */gestors/new/gestor_id/MAC..sensors..sample_time*, on sensors serà una cadena de caràcters amb el següent format: `1Tipus&2Tipus&3Tipus&4Tipus&`, on tipus pot prendre els valors Temperatura, Llum, So, NONE per indicar no presència, o ERROR per indicar presència però comunicació fallida.

- *stop*: causa l'event *TO_STOP*. Aquest event serveix per parar el mostreig de dades. Respondrem al servidor amb la mateixa petició que per l'event *setup*. Assignem 5000 ms com a *timeout* per escoltar les següents ordres del servidor.

- *start*: causa l'event *TO_START*. Respondrem al servidor amb una petició a l'adreça */gestors/state/sampling*. Un cop hem rebut aquesta ordre, només respondrem a dos events: *GET_SAMPLE* o *TO_STOP* si arriba just després d'enviar una mostra al servidor. Això és degut al fet que posarem el mòdul WiFi en mode *deep sleep* entre mostra i mostra, per tant no rebrem cap ordre. També assignem 1ms com a valor de *timeout* per complir amb el temps de mostreig de manera precisa.

El ESP8266 té diversos modes de *sleep*, però el que utilitzarem nosaltres és el més profund, el qual permet aconseguir un consum al voltant de 20,00 µA amb les condicions adequades. La conseqüència de posar el ESP8266 en aquest mode és que s'apagarà el WiFI, per tant no ens podrem comunicar amb el servidor. Per entrar en aquest mode de funcionament, es fa servir l'ordre *AT+GSLP=temps*, on temps són els mil·lisegons que estarà dormint. Per despertar d'aquest mode, segons la documentació del ESP8266 [Tea15], hem de connectar el pin *XPD_DCDC* a *EXT_RSTB*. Això simplement provoca un flanc de pujada que activa el *reset* del mòdul. En la placa de circuit imprès sobre la que està el ESP8266, el pin *XPD_DCDC* no està connectat a *EXT_RSTB*, per tant tenim dues opcions: crear el flanc de pujada des de l'Arduino o soldar aquests dos pins. Ja que hem deixat els pins analògics lliures, n'utilitzarem un per crear aquest flanc de pujada per fer el *reset* del mòdul. En definitiva, el procés per entrar i sortir del mode *deep_sleep* serà el següent:

```
wake_wifi();
send_samples();
Serial.setTimeout(2000);
if (Serial.find("action"))
{
    Serial.readStringUntil('=');
    if (Serial.readStringUntil('&') == "stop")
    {
        automat(TO_STOP);
        break;
```

```
    }
}
deep_sleep(String(600000));
```

1) Despertar el mòdul WiFi creant un flanc de pujada en el pin de *reset* i esperar a rebre una IP del servidor. Després d'algunes proves, hem determinat que amb un temps màxim de 15 segons sempre ens tornarem a connectar.

2) Enviar les mostres dels sensors actius al servidor i comprovar si ens respon amb l'event *TO_STOP*. D'aquesta manera sabrem si hem de continuar prenent mostres o hem de passar a l'estat *SETUP*.

3) *AT+GSLP=600000*: fem que el mòdul WiFi estigui en mode *deep_sleep* durant el temps màxim possible de mostreig, ja que serem nosaltres qui el desperti per tant volem assegurar-nos que estigui tota l'estona en mode *deep_sleep*.

4.1.4. Connectors i capsa 3D

Al llarg del projecte s'han realitzat els tests de funcionament sobre una placa de prototipatge. Amb l'objectiu de facilitar les connexions entre gestors i sensors, s'ha dissenyat una capsa en 3D amb el programa OpenSCAD [Ope17], tant pel dispositiu gestor com per cada sensor. Com ja hem explicat, els dispositius sensors requereixen 4 línies de contacte amb el gestor, per tant s'ha decidit utilitzar connectors jack de 4 contactes. Per unir aquests connectors, s'ha utilitzat cable apantallat 22 AWG amb tres fils de contacte més la malla de massa.

En el cas del dispositiu gestor, s'ha agafat com a referència el disseny realitzat per Kelly Egan [Ega17]. S'ha ampliat la mida de la caixa per encabir tot el circuit que féiem servir en la *protoboard* (figura 4.4). També s'han afegit els forats de radi 3.5mm per posar les 4 femelles dels connectors jack. Per acabar, s'ha augmentat l'alçada de la capsa per tal de poder situar sense problemes el mòdul ESP8266 en una de les parets. Amb totes les modificacions, les mides de la capsa són: 9,2cm x 7,8cm x 4,2 cm. El resultat final es pot veure en la figura 4.5.

4.2. Sensors

Per simplificar el sistema, s'ha decidit que els microcontroladors dels sensors facin les operacions necessàries amb les dades que recullen dels sensors per obtenir directament el resultat que es vol guardar a la base de dades. Per aconseguir-ho, s'han hagut de realitzar programes diferents per a cada sensor però complint amb el protocol de comunicació amb el gestor, és a dir, enviar un *String* en el format *GOdades$*. Tots els programes seguiran el mateix esquema de funcionament (figura 4.6) en què recolliran la dada del sensor i l'enviaran pel port sèrie quan el gestor els hi indiqui.

Per poder complir amb l'esquema de funcionament, apagant i encenent el microcontrolador entre temps de mostreig, s'ha de modificar el *bootloader* de la placa Digispark, la qual utilitza Micronucleus [Tim17a]. Per defecte, quan s'alimenta la placa, hi ha un interval de 5 segons en què es detecta si el USB està connectat per programar el microcontrolador. Per eliminar aquest temps d'espera, s'ha de carregar un *bootloader* modificat, el qual en comptes d'esperar 5 segons, llegeix el pin 0 de la placa, i si està en estat baix, permet carregar un nou programa. El *bootloader* utilitzat es pot trobar a [Tim17b].

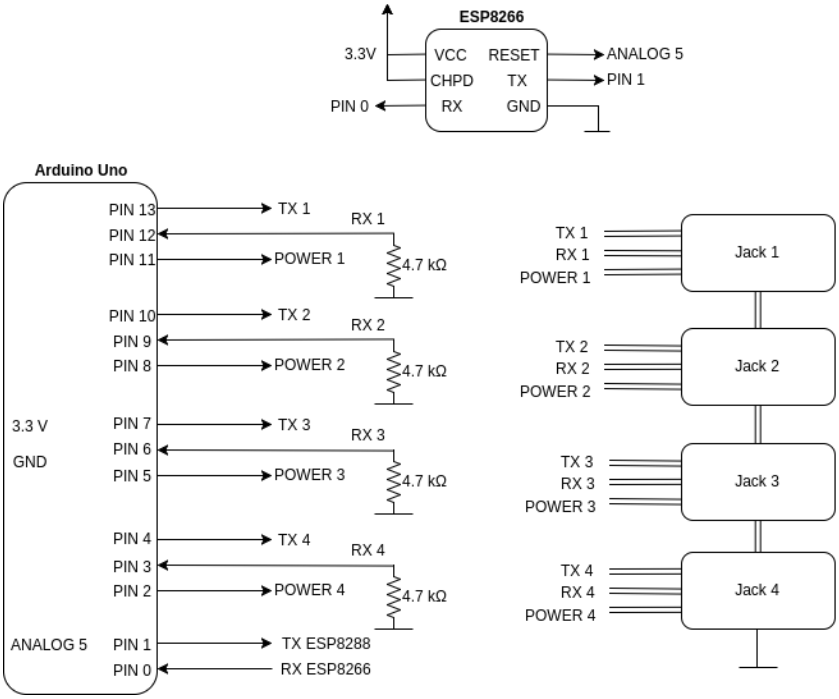

Figura 4.4.: Circuit del dispositiu gestor.

Per acabar, s'han fet modificacions amb l'objectiu d'estalviar energia. Hem desactivat elements del microcontrolador que no necessitem: Watchdog, ADC (excepte en el cas del micròfon), i comparador analògic. També hem fet que els microcontroladors funcionessin amb el *clock* a 8MHz en comptes dels 16MHz que venen per defecte. El resultat d'aquestes modificacions es pot veure en l'apartat 5.

4.2.1. DS18B20

El sensor de temperatura DS18B20 no requereix calibració i utilitza el protocol *One Wire*, en què només es necessita un pin per llegir les dades del sensor i alimentar-lo. És un protocol *half-duplex*, d'esquema mestre-esclau, en què diversos dispositius es poden connectar en el mateix bus de dades/alimentació (figura 4.7), ja que cada dispositiu està identificat amb un identificador de 64 bits gravat en una memòria ROM (Read-Only Memory).

Per alimentar el sensor podem seguir el model de la figura 4.7, anomenat *parasite power mode*, o podem alimentar el sensor amb un pin extra, mantenint la connexió amb una resistència de 4,70 kΩ entre el pin de dades i el d'alimentació. En el nostre cas seguirem aquest últim mètode, connectant directament l'alimentació del sensor a l'alimentació del microcontrolador tal com es pot observar en la figura 4.8.

Per poder llegir la temperatura, primer es va intentar utilitzar un programa en què no es requeria saber l'adreça del sensor, ja que es buscava cada cop abans de fer una lectura. A causa del fet que aquest programa excedia la memòria de programa disponible, hem utilitzat un programa per llegir l'adreça i un cop la tenim podem simplificar el programa per llegir la temperatura, adreçant-nos directament al sensor. El procés de lectura és el següent:

1) Iniciem el procés de conversió enviant l'ordre *Convert T* (0x44) al sensor amb identifica-

Figura 4.5.: Capsa 3D que conté el dispositiu gestor.

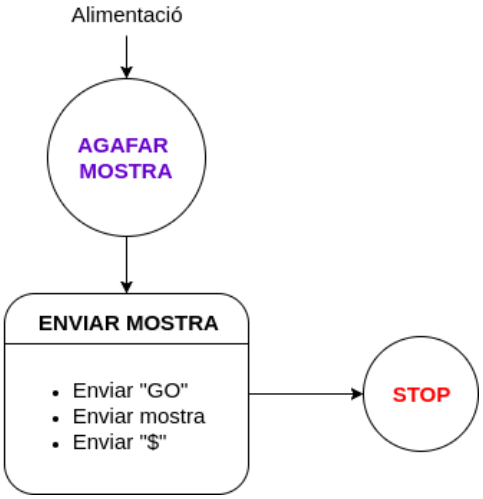

Figura 4.6.: Esquema de funcionament dels controladors dels sensors.

dor `0x28 0x14 0x24 0x30 0x09 0x00 0x00 0xB8`. El sensor llegirà el seu ADC intern i n'escriurà el resultat en uns registres anomenats *Scratchpads*. Aquest procés pot tardar entre 94ms, resolució de 9 bits, i 750ms, resolució de 12 bits. S'ha decidit esperar 1 segon a llegir el resultat per tal de que les lectures al sensor tardin el mateix, fent que l'interval de mostreig sigui més exacte.

2) Llegim els 9 bytes del registre *Scratchpad* enviant l'ordre 0xBE. D'aquests 9 bytes, els dos primers són la temperatura i els altres contenen configuracions del sensor i un byte de CRC (Cyclic Redundancy Check). S'ha decidit utilitzar la resolució de 12 bits (0,06 °C).

3) Els dos bytes que representen la temperatura estan guardats en 16 bits amb signe estès. Per tant els quatre primers bits indiquen si es tracta d'una temperatura positiva o negativa, i els 12 restants són el valor de la lectura. Per obtenir el valor en °C, multipliquem per 0.0625 el valor llegit. Si la temperatura és negativa, haurem de fer el complement a 2 abans de la multiplicació. Dividim per 100 per obtenir la part entera de la lectura i realitzem l'operació mòdul entre 100 per tenir la part decimal.

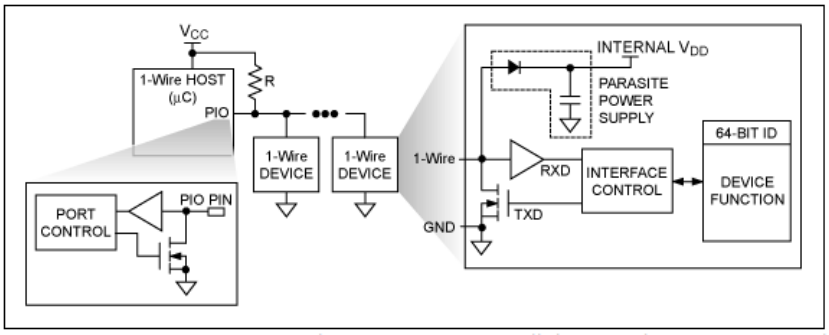

Figura 4.7.: Connexions en una comunicació *One Wire* [Int17a].

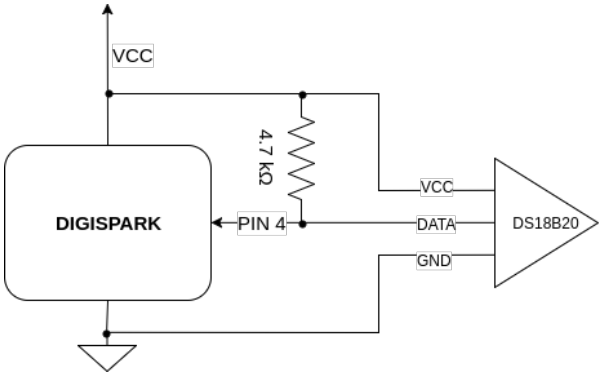

Figura 4.8.: Esquema del circuit per llegir el sensor DS18B20.

```
LowByte = data[0];
HighByte = data[1];
TReading = (HighByte << 8) + LowByte;
SignBit = TReading & 0x8000;

if (SignBit)
{
    TReading = (TReading ^ 0xffff) + 1;
}
Tc_100 = (6*TReading)+TReading/4;

Whole = Tc_100 / 100;
Fract = Tc_100 % 100;
```

Després de tot aquest procés podem enviar la temperatura en forma de *String* a través del port sèrie.

4.2.2. TSL2591

Per utilitzar el sensor de llum TSL2591 s'ha de dissenyar primer una placa de circuit imprès a causa de la mida del sensor: 2mm x 2.4mm x 0.65mm. Seguint les mides que suggereix

la documentació del sensor [AG15], s'ha dissenyat una placa de circuit imprès utilitzant el software KiCad. Tal com es pot veure en la figura 4.9, s'han fet les pistes per soldar el sensor amb les mides indicades: amplada de 0.35 mm, llargada de 1.2 mm, i separació entre pistes de 0.65 mm. Després hem afegit els connectors per soldar pins a la placa i poder-la fer servir sobre la *protoboard*.

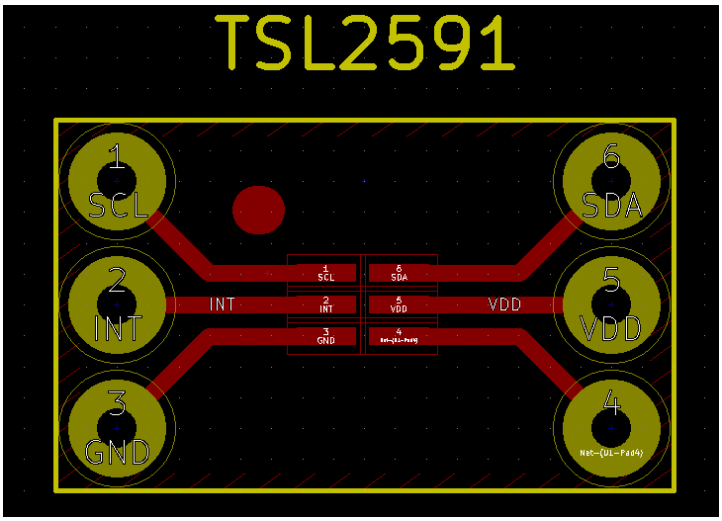

Figura 4.9.: Esquema placa circuit imprès pel sensor TSL2591.

Un cop tenim el sensor soldat sobre la placa (figura 4.10), podem procedir a connectar-lo al microcontrolador seguint les instruccions de la documentació. Amb el circuit 4.11 hem provat de llegir les mostres a través del protocol I2C, seguint els mateixos passos que es realitzen en la llibreria per la placa d'Adafruit [Ada17]:

1) Inicialitzem el protocol I2C, indicant que el microcontrolador actuarà com a *master*.

2) Iniciem una transmissió de dades amb l'esclau amb adreça 0x29, que segons la documentació es la del sensor.

3) Escrivim en el registre 0x14, el qual correspon al byte baix del ADC0 del sensor.

4) Demanem 2 bytes de dades a l'esclau amb adreça 0x29.

5) Llegim els dos bytes i els combinem en una variable de 16 bits.

En aquest punt hauríem de tenir la mostra del sensor a punt per llegir, però al fer-ho observem que té el valor 65535. Això és degut al fet que tots els bits de la variable mostra són 1 a causa que el sensor no ha posat en estat baix la línia de dades, indicant que no ha rebut les nostres ordres. Hem comprovat l'alimentació del sensor, 3.3V, les connexions del circuit, l'execució del programa llegint d'un altre microcontrolador configurat com a esclau, i no s'ha pogut trobar la causa del problema.

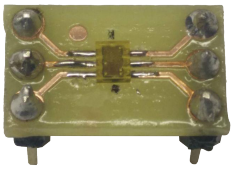

Figura 4.10.: Placa de circuit imprès amb el sensor TSL2591 soldat.

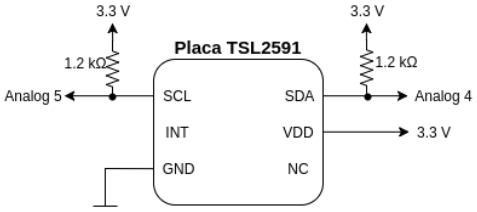

Figura 4.11.: Circuit de test per comunicar-nos amb el TSL2591.

4.2.3. Micròfon Electret

El micròfon utilitzat està inclòs en una placa de circuit imprès amb un circuit amplificador (4.12) el qual utilitza un amplificador operacional *rail-to-rail* per aconseguir un guany al voltant de 60V/V.

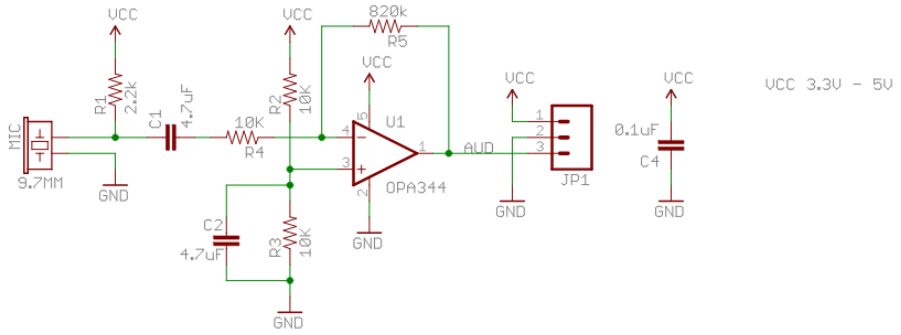

Figura 4.12.: Esquemàtic de la placa de circuit imprès del micròfon electret [Spa17b].

La sortida de la placa ens dóna un valor analògic el qual haurem de convertir a digital amb el convertidor ADC de 10 bits del microcontrolador ATTiny85. En situació de silenci, hauríem de llegir idealment valors al voltant de 512 (la meitat del valor màxim). Per calcular un valor de referència de soroll ambient, usarem la següent fórmula, on `x` és el valor llegit del ADC i `m` és la mitjana de valors:

$$\frac{1}{N}\sum_{n=1}^{N} x_n^2 - m^2$$

Durant els tests realitzats, s'ha observat que el micròfon tarda un temps variable a oferir

valors acceptables, per tant s'ha decidit executar dues vegades la funció de càlcul amb un període de 2 segons amb un temps de mostreig de 100ms i enviar al servidor la dada de la segona lectura. També s'ha pogut observar que aquest dispositiu és molt sensible als altres elements connectats al circuit, fent que es llegeixin valors totalment erronis, com es pot veure en l'apartat 5.

4.2.4. Capsa 3D

Pels dispositius sensors s'han realitzat dos dissenys lleugerament diferents. Els dos estan basats en el disseny realitzat per Jan-Willem [Jan17]. Igual que en el cas anterior, hem ampliat la mida de la capsa per introduir les connexions necessàries (figura 4.13) i en aquest cas hem eliminat les obertures pels pins de la part superior. Les mides finals de la capsa són: 3,4cm x 5,3cm x 2,2cm, forats de muntura amb radi de 2mm. En la capsa pel micròfon electret, s'ha fet un forat en la part superior, amb un radi de 5 mm, per fer sortir el micròfon, enganxant la placa en la part interior de la capsa (figura 4.14). En el cas del sensor de temperatura s'ha decidit deixar la capsa oberta per utilitzar-la com a mostra per veure les connexions fetes a l'interior.

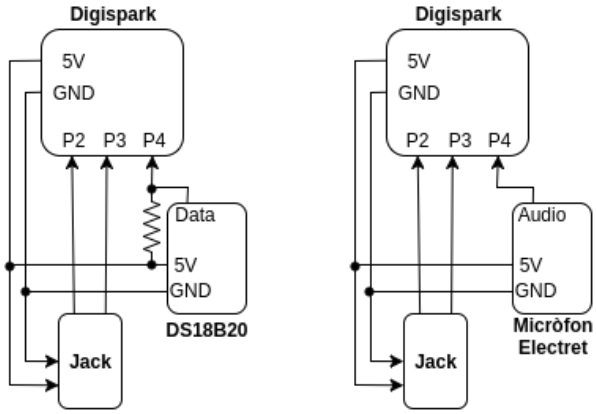

Figura 4.13.: Circuits pels sensors de temperatura (esquerra) i el micròfon electret (dreta).

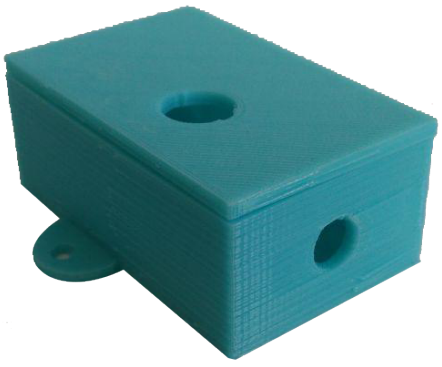

Figura 4.14.: Capsa 3D pel micròfon electret.

4.3. Servidor i Punt d'accés

L'última part del sistema és la Raspberry Pi, que ens proporcionarà una xarxa WiFi tant per permetre a l'usuari accés al control dels sensors i a les seves dades, com per proporcionar el canal a través del qual els gestors podran interactuar amb el servidor. L'objectiu d'utilitzar la Raspberry com a punt d'accés és assegurar que el nostre sistema pot funcionar en qualsevol entorn, ja que són els mateixos dispositius del sistema que s'encarreguen de crear els mitjans de comunicació. A més a més, gràcies a que disposem de dues interfícies en la Raspberry, *ethernet* i WiFi, podem oferir l'opció d'accedir a la interfície de control del sensors des d'un punt extern a la xarxa WiFi creada. Això pot ser útil per controlar els sensors o recollir dades a distància. Per aconseguir aquestes funcionalitats, haurem de fer ús de diferents eines (figura 4.15):

- *Hostapd*: és un programa que gestiona la creació del punt d'accés WiFi, l'autentificació dels clients, etc. A la web oficial [Mal13] es pot trobar informació sobre quins paràmetres i com es poden configurar. L'utilitzarem per crear el punt d'accés que utilitzaran els gestors i el client.

- *Dnsmasq*: proporciona serveis de DNS (Domain Name Server) i DHCP (Dynamic Host Configuration Protocol). Està dissenyat per utilitzar pocs recursos i l'utilitzarem per assignar IPs estàtiques als gestors i donar un nom a la web de gestió. A la web oficial [Kel17] trobem informació de com aconseguir-ho.

- *Flask*: serà el software que utilitzarem per escriure el programa principal que s'encarregarà de la gestió de la base de dades, comunicacions amb l'usuari i els gestors, i implementació de la màquina d'estats del servidor.

- *MongoDB*: serà la base de dades que utilitzarem tant per guardar les dades recollides dels sensors com per mantenir un estat global del sistema.

- *Docker*: l'utilitzarem per gestionar els dos processos principals: servidor Flask i la base de dades MongoDB. També ens serà útil durant tot el procés de desenvolupament per facilitar i simplificar la feina per les raons que hem exposat en l'apartat 3.5.4.

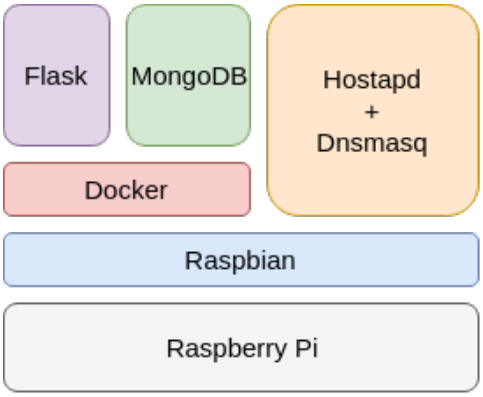

Figura 4.15.: Software utilitzat per cada funcionalitat.

4.3.1. Docker

Normalment s'utilitza Docker per tenir un entorn similar durant el desenvolupament i el desplegament de l'aplicació, però en el nostre cas hem de considerar que el desplegament es farà en un sistema operatiu de 32 bits (Raspbian), per tant haurem de tenir dos `Dockerfiles` diferents, per 64 bits i 32 bits, que utilitzin la mateixa configuració i els paquets instal·lats siguin el més semblant possible. En total, tindrem quatre imatges:

- Flask (64 bits): basarem aquesta imatge en el sistema operatiu Alpine Linux, el qual ocupa molt poc espai (60 MB). Instal·larem el paquet PIP, que és un gestor de paquets de python que ens permetrà instal·lar Flask, PyMongo (API per interactuar amb MongoDB des de Python), i Requests (llibreria per realitzar peticions HTTP. Amb l'ordre *WORKDIR* indicarem que comparteixi el directori on tenim el codi font de l'aplicació web amb el sistema *host*. Per acabar, exposem el port 80 per oferir el servei web.

- MongoDB (64 bits): utilitzarem directament la imatge oficial que proporciona MongoDB a través de DockerHub, concretament la versió 3.4. L'única modificació que farem és indicar que les dades les guardi en un directori de la màquina *host*, per tal que no s'esborrin cada cop que parem la màquina.

- Flask (32 bits): l'única diferència respecte a l'imatge de 64 bits és que aquest la basarem en el sistema operatiu Raspbian, la resta és completament igual.

- MongoDB (32 bits): aquesta és la imatge que ha costat més de crear degut a les raons que expliquem en l'apartat 4.3.3. Ens hem basat en un Dockerfile [btn17] que ens ofereix la versió 2.6, introduint una lleugera variació: activar el *journaling*. El *journaling* s'utilitza per crear punts de partida en la base de dades per tal de recuperar-se d'una parada inesperada. Activar aquesta opció és vital si tenim en compte que per parar la Raspberry l'usuari no pot aturar els diferents serveis que hi corren de manera correcta, per tant si no activem el *journaling* segurament corromprem la base de dades, cosa que ha succeït múltiples vegades durant els tests que s'han realitzat.

Per gestionar l'encesa i parada de les imatges, s'utilitza Docker Compose. En un fitxer de text a part dels `Dockerfiles`, `docker-compose.yml`, s'indiquen tots els paràmetres que hem exposat anteriorment. A més a més, utilitzant l'ordre *link*, creem un punt de comunicació entre els dos *containers* que ens permetrà actuar sobre la base de dades des de l'aplicació web, tal com es veu en la figura 4.16. Amb els fitxers de configuració creats, l'únic que hem de fer per activar els serveis és executar la comanda `docker-compose up`. Un cop s'ha executat, s'iniciarà automàticament cada cop que s'encengui la Raspberry ja que ho indiquem amb l'ordre *restart:always* dins el fitxer `docker-compose.yml`.

4.3.2. Aplicació Web

L'aplicació web és la part més important del sistema perquè és la que el convertirà en un sistema accessible a qualsevol usuari. A través d'ella podrem configurar els gestors tant de manera automàtica com manual, gestionar la recollida de dades, descarregar períodes de dades, etc. El disseny de l'aplicació s'ha separat en dues parts: la interfície per l'usuari, i l'API web pels gestors. En la primera part ens hem centrat en l'objectiu de crear un sistema que pugui ser utilitzat per qualsevol tipus d'usuari, per tant la interfície per interactuar amb els sensors

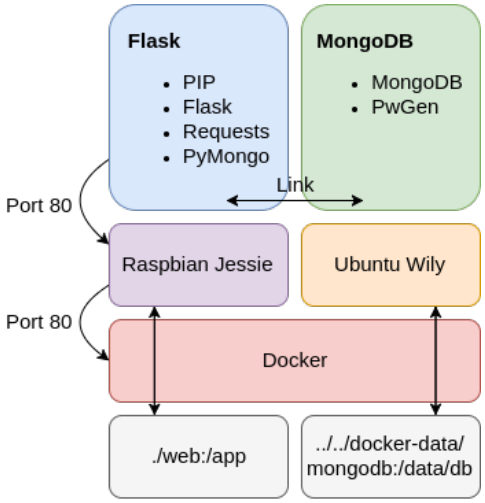

Figura 4.16.: Esquema dels containers Flask i MongoDB.

ha de ser el més entenedor possible, oferint suficients funcionalitats però d'una manera senzilla. En l'altra part hem volgut ressaltar la qualitat de ser un sistema fàcilment ampliable, oferint una API clara i concisa que permeti afegir elements de manera senzilla.

Igual que en els gestors, l'aplicació web també passa a través d'estats depenent dels events però en aquest cas podem diferenciar dues màquines d'estats:

- *Sessions*: aquesta màquina d'estats serà amb la que interactui l'usuari. A través de la interfície web, podrà generar events per canviar d'estat, com per exemple iniciar la recollida de mostres, activar o desactivar sessions, etc. En l'apartat 4.3.2 l'expliquem en detall.

- *Gestors*: per configurar els gestors hem de tenir constància de si reben les nostres ordres, per tant ens caldrà definir uns estats que dependran de les accions de l'usuari i de la resposta que es rebi des dels gestors. Aquesta màquina d'estats estarà englobada dins l'anterior, ja que s'utilitzarà com a guia visual perquè l'usuari pugui saber en quina situació es troba cada dispositiu però condicionarà el funcionament del sistema.

Per estructurar el codi font de l'aplicació s'ha seguit un model molt comú en les aplicacions fetes amb Flask (figura 4.17). Dintre la carpeta *Templates* tenim tots els fitxers HTML que seran processats pel llenguatge Jinja2. Una de les funcionalitats que ens ofereix aquest llenguatge és poder definir un fitxer com a base pels altres, creant un aspecte constant a través de totes les pàgines per les quals pot navegar l'usuari. Utilitzarem el fitxer *base.html*, on carregarem els fitxers d'estil (CSS) i definirem el mateix encapçalament i peu de pàgina per tots els altres fitxers, fent que només es modifiqui l'espai central de la pàgina. Com que es tracta d'un sistema que no requereix connexió a Internet, servirem els fitxers d'estil, imatges, fonts, i scripts directament de la Raspberry Pi, els quals estaran tots dintre la carpeta *Static*. Per acabar, hem separat el codi Python en diferents mòduls:

- *web.py*: és el fitxer principal de l'aplicació web, encarregat de començar el servidor en

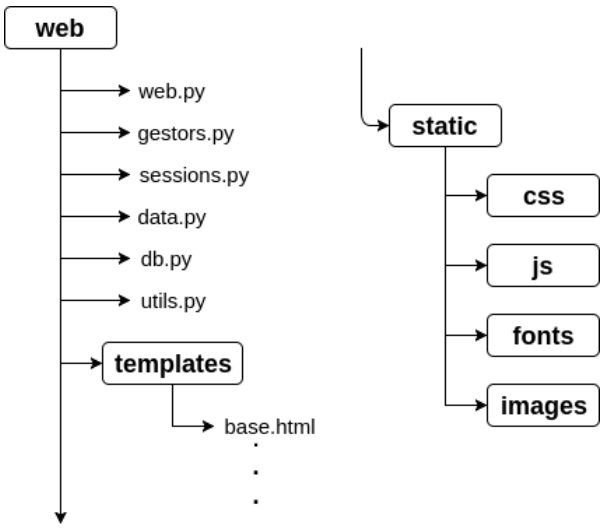

Figura 4.17.: Estructura de fitxers de l'aplicació web.

la direcció IP 0.0.0.0 (totes les interfícies), pel port 80. Gestiona totes les peticions que comencin amb /.

- *gestors.py*: conté totes les funcions encarregades d'interactuar amb la màquina d'estats dels gestors i de recollir les dades. Forma part directa de l'aplicació Flask i gestiona totes les peticions que comencin amb /gestors.
- *sessions.py*: conté totes les funcions encarregades d'interactuar amb la màquina d'estats del Sistema. Forma part directa de l'aplicació Flask i gestiona totes les peticions que comencin amb /sessions.
- *data.py*: s'utilitza per gestionar la descàrrega de dades. Forma part directa de l'aplicació Flask i gestiona totes les peticions que comencin amb /data.
- *db.py*: conté totes les funcions necessàries per interactuar amb la base de dades. No forma part directa de l'aplicació Flask.
- *utils.py*: conté funcions de caràcter general, utilitzades per altres mòduls. No forma part directa de l'aplicació Flask.

Interfície per l'usuari

A l'hora de dissenyar la seqüència que s'ha de seguir des del moment que s'inicialitza el sistema fins que es volen recollir les dades (figura 4.18), s'ha volgut agafar com a base la idea de poder tenir diferents desplegaments de sensors guardats en el que anomenem *sessions*. Aquestes sessions volen representar diferents escenaris en què es facin mesures amb l'objectiu que l'usuari pugui tenir un registre de la configuració que s'han realitzat les recollides de dades. A més a més, també es vol donar l'opció de poder crear múltiples sessions de tal manera que el pròxim cop que s'engegui el sistema es carregui automàticament en els gestors la configuració de la sessió activa, permetent tenir múltiples configuracions de sensors sense fer-ne el desplegament físic.

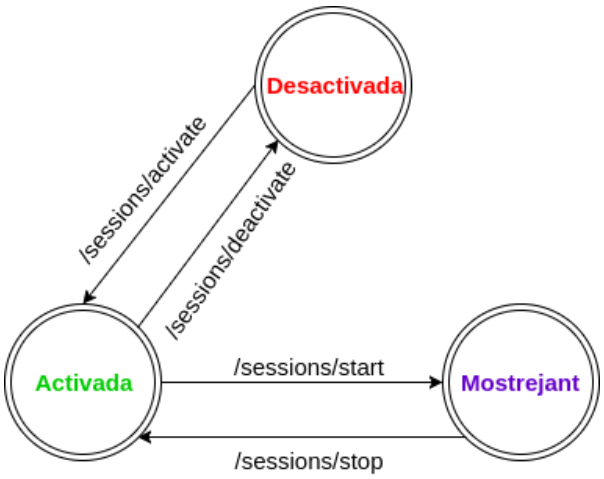

Figura 4.18.: Màquina d'estats de les sessions.

A la pàgina principal de la web (figura 4.19) es poden crear, editar, i eliminar sessions. També es poden observar els paràmetres de cada sessió: identificador, lloc, i si està activada. Només pot estar activada una sessió a la vegada, ja que aquest paràmetre l'utilitzarem per saber quina configuració s'ha de carregar als gestors i a on s'han de guardar les dades. Un cop s'ha activat una sessió, prement el botó *Activar Sessió*, podem procedir a configurar els gestors.

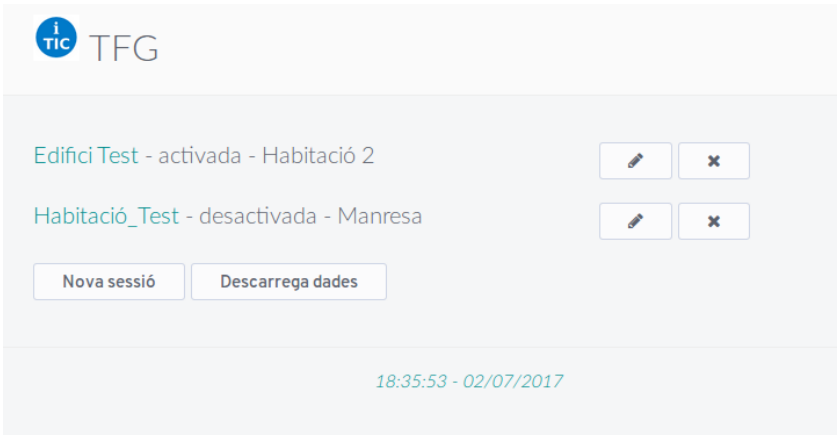

Figura 4.19.: Pàgina principal per gestionar les sessions.

Tots els gestors que es connectin aniran apareixent a la pàgina de la sessió activa (figura 4.20). Si ja existeix una configuració pel gestor, se li carregarà automàticament. En cas contrari, apareixerà amb un identificador *undef*, indicant que requereix configurar-lo. Com es pot observar en la figura 4.20, aquesta pàgina serveix com a panell de control de les mesures que es volen fer, ja que des d'aquí es poden parar, configurar, i engegar (començar les mesures) els gestors. També es pot tenir una vista general d'en quin estat està el sistema, i en particular cada gestor, utilitzant el següent sistema de colors:

- *Vermell*: indica que la configuració que es veu en pantalla no coincideix amb la que està

en el gestor.

- *Verd*: indica que el gestor té els mateixos paràmetres que es poden observar a la pantalla.
- *Morat*: indica que el gestor està mostrejant.

Per interaccionar amb els gestors, s'utilitzen els botons situats a la part inferior de la pàgina:

- *Enviar configuració*: fa una petició GET a l'URL */gestors/send*. L'utilitzarem per enviar la configuració que apareix en pantalla a tots els gestors a través . Si el gestor ho rep, el seu identificador passarà de color vermell a verd.
- *Scan*: fa una petició GET a l'URL */sessions/scan*. L'objectiu d'aquesta ordre és demanar als gestors que ens enviïn els seus paràmetres i sobreescriure el que apareix a la web. D'aquesta manera es pot detectar si hem connectat un sensor. Si el gestor no respon, el seu identificador apareixerà de color vermell, en cas contrari apareixerà de color verd.
- *Començar*: fa una petició GET a l'URL */sessions/start*. Els gestors que estiguin de color verd, se'ls hi enviarà l'ordre per començar a agafar mostres. Si la reben, passaran al color morat, en cas contrari no canviaran de color. també provocarà que la sessió passi a l'estat mostrejant, fent que els següents gestors que es connectin passin directament a l'estat de mostreig si ja tenen una configuració a la web i bloquejant l'opció de desactivar la sessió.
- *Parar*: realitza una petició GET a l'URL */sessions/stop*. Com ja hem explicat en l'apartat 4.1.3, els gestors rebran aquesta ordre just després d'enviar una mostra al servidor. També provocarà que la sessió passi de l'estat mostrejant a activada.

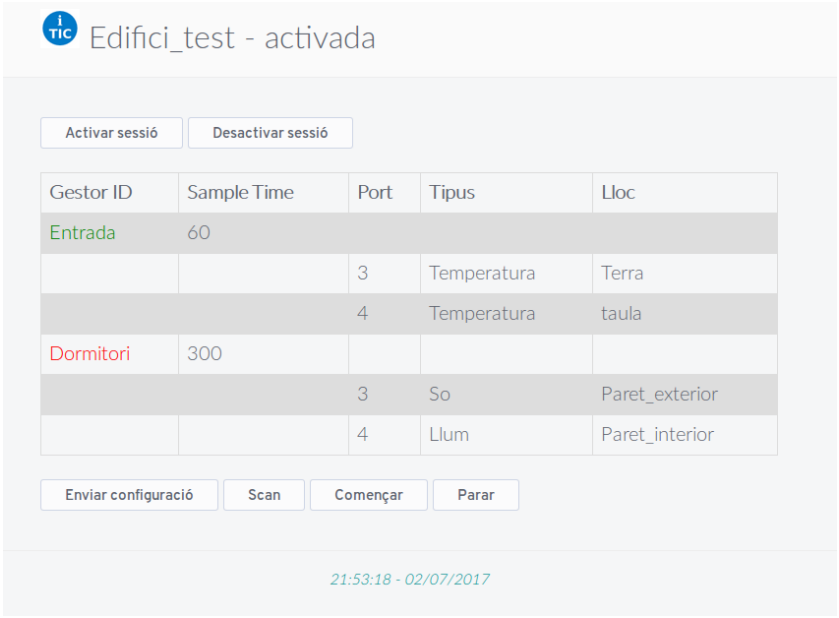

Figura 4.20.: Pàgina principal per controlar el sistema.

Per configurar un gestor, es pot accedir a la pàgina de configuració fent clic sobre l'identificador del gestor. En la figura 4.21, veiem tots els camps que es poden configurar: identificador, temps de mostreig a triar a entre 60, 120, 300 i 600 segons, i per acabar el lloc on està situat el sensor. Un cop es fa clic al botó *Desa*, s'envia el formulari a l'adreça */gestors/setup*, actualitzant només els camps on s'ha escrit alguna cosa, els altres mantenen el valor anterior. Per esborrar el gestor, es pot fer clic al botó *Esborrar gestor* i llavors apareixarà un quadrat de text demanant que confirmem l'acció, si acceptem es farà una petició POST a */gestors/delete*, esborrant el gestor de la base de dades. A la capçalera de la pàgina, veiem que surt l'adreça MAC del gestor. L'objectiu d'això és proporcionar a l'usuari una manera d'identificar de manera única els seus dispositius.

Figura 4.21.: Pàgina per configurar un gestor.

Per acabar, l'última part on pot accedir l'usuari és a la descàrrega de dades, fent clic al botó *Descarrega dades* de la pàgina principal (figura 4.19). Aquest botó farà una petició GET a l'adreça */data/list-data*. Com es pot veure en la figura 4.22, obtindrem una taula mostrant totes les dades disponibles per descarregar. Tot i que esborrem totes les sessions, les dades no s'esborraran, ja que estan en llocs diferents de la base de dades, tal com expliquem en l'apartat 4.3.3. Fent clic sobre el tipus de sensor del qual volem descarregar les dades anirem a parar a la pàgina que es pot veure en la figura 4.23. Per identificar de manera única cada sensor, primer hem de triar el paràmetre *lloc* del desplegable de la part superior. Un cop tenim identificat de manera única el sensor del qual volem descarregar les dades, tenim dues opcions: consultar dades durant un rang de temps o obtenir totes les dades disponibles. Si comencem a escriure en un dels requadres per triar un rang de temps, es desplegarà un menú amb totes les hores en les quals tenim dades i que coincideixi amb el que estem escrivint. Això ho aconseguim amb una funció de Javascript, més concretament del *framework* jQuery, la qual carrega una llista de valors amb les hores fent una petició al servidor. Quan detecta que s'escriu en el requadre, executa la funció *autocomplete* comparant el que s'està escrivint amb la llista

de valors que es té. Si no triem un dels valors del desplegable, hem de vigilar a complir amb el format de data que s'especifica. Quan fem clic a la icona de descarregar, obtindrem un fitxer en format CSV, amb el següent format: dada,HH:MM:SS. El procés per obtenir aquest fitxer està explicat en l'apartat 4.3.3.

Figura 4.22.: Pàgina mostrant les dades disponibles.

API Web

L'objectiu principal de l'API (Application Programming Interface) web és proporcionar un fil de comunicació entre l'usuari i els gestors. Ens ha de permetre conèixer en quin estat estan en tot moment i si estan configurats o no. També ha de servir com a canal de comunicació entre els sensors finals i la base de dades, permetent l'identificació única de les mostres.

Per poder-nos comunicar amb els gestors, necessitem una identificació única per cadascun d'ells. Aprofitant que cada gestor té un ESP8266, utilitzarem l'adreça MAC com a identificador dels gestors. Això no només ens permetrà identificar-los en la banda del servidor, sinó que també els podrem identificar físicament. Un cop el gestor s'hagi connectat al punt d'accés, sempre tindrà la mateixa IP a causa de la configuració que expliquem en l'apartat 4.3.4, per tant també ens hi podrem referir utilitzant la seva IP.

El procés per configurar un gestor es pot veure en la figura 4.24. Les peticions que enviem als gestors les creem utilitzant la llibreria Requests [Rei17] i estan al fitxer `utils.py`. Un cop hem enviat una ordre al gestor, esperem que ell faci una petició GET a l'URL */gestors/state/gestor_id/estat*, on estat és una variable que utilitzarem per actualitzar l'estat del gestor tal com hem explicat en l'apartat 4.3.2. D'aquesta manera complim amb l'objectiu de

Figura 4.23.: Pàgina per configurar la descàrrega.

mantenir a l'usuari informat d'en quin estat estan els gestors. Per aconseguir que els gestors es puguin configurar automàticament, quan rebem la petició inicial a */gestors/new/...*, comprovem a la base de dades si ja existeix una configuració per la MAC. Si ja existeix, farem dues coses: primer combinarem els paràmetres dels sensors antics amb els nous per lliurar a l'usuari d'haver de tornar-los a configurar, després mirarem en quin estat està la sessió, si està mostrejant, enviarem l'ordre de començar a mostrejar al gestor. D'aquesta manera, si algun gestor es para per qualsevol motiu, només cal tornar-lo a engegar per tal que s'incorpori ell mateix en el sistema.

Per emmagatzemar les dades dels sensors oferirem l'adreça */gestors/data/gestor_id/sensor_port..count/data*. Utilitzarem gestor_id per identificar el gestor segons la sessió activa i sensor_port per saber de quin sensor exacte es tracta. El motiu del paràmetre count està explicat en l'aparat 4.3.3. Si l'estat de la sessió no és mostrejant, enviarem l'ordre de parada al gestor. Per altra banda, si la funció per inserir dades a la BD, ens retorna un valor buit, voldrà dir que per alguna raó no hem rebut la mostra 0, la qual marca el començament d'un paquet de dades d'una hora, per tant enviarem l'ordre de parada al gestor i després l'ordre de mostrejar, per fer un *reset* sense requerir la intervenció de l'usuari.

4.3.3. MongoDB

En la base de dades guardarem les configuracions dels gestors i els seus sensors, l'estat de les sessions, l'estat dels gestors, i les dades recollides de cada sensor. L'esquema de la base de dades està separat en dues col·leccions, dades i sessions, per tal de mantenir un registre de totes les dades independentment de la gestió que es faci de les sessions.

Per identificar les sessions de manera única utilitzarem el camp *sessio_id*. Quan creem una nova sessió comprovarem en la banda del servidor que aquest paràmetre no estigui repetit en

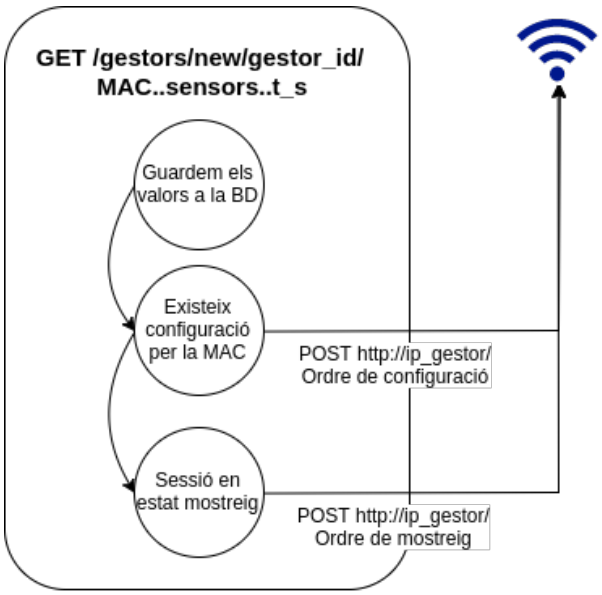

Figura 4.24.: Procés de configuració d'un gestor.

la BD. Per modificar l'estat de la sessió realitzarem una operació d'actualització i, ja que en MongoDB la clau primària és un identificador que ens proporciona ell, podrem modificar qualsevol camp dels documents sessió. Dintre d'aquests documents de sessió tenim el camp gestors, el qual és una llista de documents. Aquesta estructura ens permetrà realitzar operacions de consulta més ràpidament, ja que es filtrarà la cerca pel camp sessio_id. Per afegir i eliminar documents dins la llista de gestors, en comptes de realitzar operacions d'esborrar o afegir directament, usarem els operadors *$pull* i *$push* de l'operació de modificació perquè ens permeten treballar de manera més senzilla amb les llistes de documents. Dintre el document de gestors tornem a tenir una llista de documents pels sensors. Ja que afegir els sensors d'un en un seria molt poc eficaç, el que farem és inserir una llista de documents amb els 4 sensors possibles i quan els gestors ens diguin que tenen un cert sensor connectat en un port, actualitzarem el document que correspongui modificant el camp state posant-lo a active.

```
{
    'sessio_id': sessio_id,
    'lloc': lloc,
    'state': estat,
    'gestors':
    [{
        'id': gestor_id,
        'ip': gestor_ip,
        'sample_time': sample_time,
        'state': estat,
        'MAC': MAC,
        'sensors': [{'port': port, 'state': estat,
                     'type': tipus, 'lloc: lloc}, ...]
    },
```

```
    ...]
}
```

Per emmagatzemar les dades dels sensors hem seguit l'esquema proposat per MongoDB [Run14], en el qual les dades de sensors es guarden en paquets de certa durada. En el nostre cas hem triat utilitzar paquets d'1 hora. L'objectiu d'aquesta estratègia és poder tenir una sèrie de temps estalviant-nos temps de lectura i espai que ocupa la base de dades. Ja que MongoDB no obliga a seguir un esquema a l'hora de guardar dades, la mida del paquet de dades serà variable en funció del temps de mostreig. L'únic desavantatge és que hem de forçar a triar un temps de mostreig que sigui divisor de 60, ja que si no els paquets de dades quedarien descompensats. Aquests paquets seran identificats de manera única utilitzant els següents camps: sessio_id, gestor_id, tipus de sensor, i lloc.

```
{
    'sessio_id': sessio_id,
    'gestor_id': gestor_id,
    'type': tipus de sensor,
    'hour': hora del paquet,
    'data':
    {
        '0': mostra,
        '1': mostra,
        ...
        '60/temps_mostreig - 1': mostra
    }
}
```

Per inserir les dades a la base de dades seguirem una estratègia que prioritzi l'ús d'actualitzacions en comptes d'insercions. Per aconseguir això el gestor ens ha d'indicar amb un nombre quina mostra ens envia. Si es tracta de la mostra 0, inserirem un paquet de dades sencer a la base de dades, posant 'x' com a valor de les mostres. A mesura que el gestor ens enviï les següents mostres, actualitzarem el valor utilitzant el nombre que ens envia el gestor com a referència. Si per alguna raó el servidor deixa de rebre les mostres, mantindran el valor 'x', i quan torni a rebre les mostres continuarà afegint-les al lloc que els hi pertoca, a no ser que hagi passat més d'una hora, cosa que provocaria que el servidor fes un *reset* de l'estat del gestor, obligant-lo a enviar la mostra 0 un altre cop. Per consultar les dades, agafarem els paquets de dades en què el camp hora estigui dins el rang de temps que es demani i crearem una marca de temps per cada mostra de manera dinàmica. Utilitzant la llibreria *datetime* de Python, agafarem l'hora del paquet com a base i li sumarem increments de temps de mostreig per cada mostra. Si alguna mostra encara te el valor 'x', la descartarem.

L'inconvenient més gran a l'hora d'utilitzar MongoDB en el nostre sistema és el fet que no s'ofereix suport per les versions més recents pels sistemes operatius de 32 bits. A més a més, la mida de la base de dades està limitada a 2 GB. En el nostre cas aquest no és un aspecte molt greu, ja que un paquet d'una hora amb temps de mostreig de 60 segons, el que ocupa més, ocupa al voltant de 2 kB, segons la funció *stats* de MongoDB. Dividint 2 GB entre 2 kB, obtenim que podem guardar 1.000.000 paquets (hores). Si tenim un desplegament de 20 sensors, això significa que podrem tenir-lo mostrejant durant 5,7 anys.

4.3.4. Punt d'accés, DNS i DHCP

Per permetre que tots els elements del sistema es puguin comunicar entre ells, hem de crear un punt d'accés WiFi a través de la Raspberry. Per aconseguir-ho seguirem la guia de Phil Martin [Mar17] afegint lleugeres modificacions.

Comencem instal·lant els paquets necessaris: dnsmasq i hostapd. El primer realitzarà les funcions de DNS i DHCP, el segon crearà el punt d'accés i gestionarà les connexions dels clients. Els passos a seguir són el següents:

1) Assignem la IP `172.24.1.1/24` a l'interfície wlan0 de la Raspberry de forma estàtica modificant el fitxer `/etc/network/interfaces`.

2) Configurem el programa hostapd amb el fitxer `/etc/hostapd/hostapd.conf`. Crearem un punt d'accés per l'interfície wlan0, amb SSID TFG-AP i contrasenya serra1234.

3) Configurem el programa dnsmasq amb el fitxer `/etc/dnsmasq.conf`. Indicarem que escolti només en l'adreça `172.24.1.1` de wlan0. El rang d'adreces que podrà donar és entre `172.24.1.2` i `172.24.1.254` i tindran un temps de vida infinit, per assegurar que quan els gestors es connectin sempre li donem la mateixa adreça IP. Les peticions de resolució de noms les redirigirem al servidor de DNS de Google, `8.8.8.8`, amb excepció del nom tfg.tic, el qual indicarem que el resolgui de manera local agafant els valors del fitxer `/etc/hosts`.

4) En el fitxer `/etc/hosts` indiquem que el nom tfg correspon a la IP `172.24.1.1`.

5) Activem el reencaminament de ipv4 a través de eth0. Això ens permetrà tenir accés a Internet connectats a través del punt d'accés.

Tenir accés a internet a través del punt d'accés ens servirà per si volem controlar el sistema des d'algun punt extern, ja que l'aplicació Flask escolta en totes les interfícies, només ens caldrà dirigir-nos a l'adreça IP de la interfície eth0. Un altre aspecte important de tenir accés a Internet és que la Raspberry sàpiga quina hora és, ja que normalment per saber-ho es fa una consulta a través d'Internet. A causa del fet que el sistema està pensat per funcionar de manera offline, hem afegit l'opció de modificar l'hora a través de l'aplicació. Si es fa click sobre l'hora que apareix en la part inferior de totes les pàgines, es pot modificar l'hora que la Raspberry agafarà com a referència. Per fer això cal que el *container* de l'aplicació Flask s'executi en mode privilegiat per poder modificar el temps del sistema operatiu de la Raspberry.

5. Validació experimental

Per comprovar el funcionament del sistema primer s'han traslladat els circuits de test de la placa de prototipatge a dins les capses 3D. Com es pot veure en la figura 5.1, s'han soldat cables en els contactes dels connectors femella cap a una placa situada sobre l'Arduino Uno, on es realitzen totes les connexions. S'ha utilitzat el següent codi de colors per identificar els contactes: groc = TX, taronja = RX, blau = alimentació, i gris = GND. A la part superior dreta de la imatge 5.1, es pot observar el mòdul WiFi, el qual s'ha col·locat del revés per aprofitar espai. Pel circuit del sensor de temperatura hem seguit la mateixa estratègia, és a dir, utilitzar una placa situada sobre el microcontrolador per realitzar totes les connexions (figura 5.2). En el cas del micròfon electret, s'ha decidit deixar el circuit sobre la placa de prototipatge (figura 5.3) a causa de problemes d'espai a l'hora de traslladar el circuit.

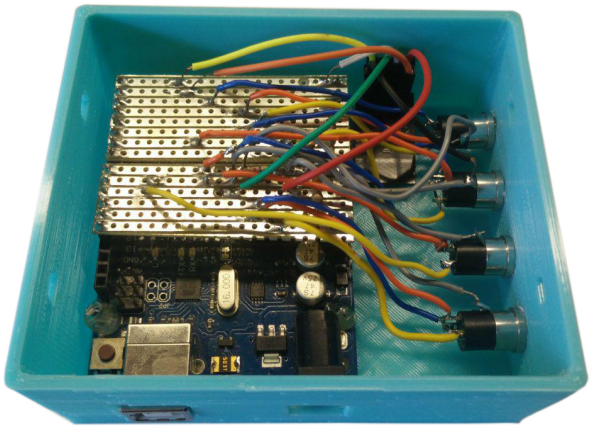

Figura 5.1.: Muntatge final del gestor dins la capsa 3D.

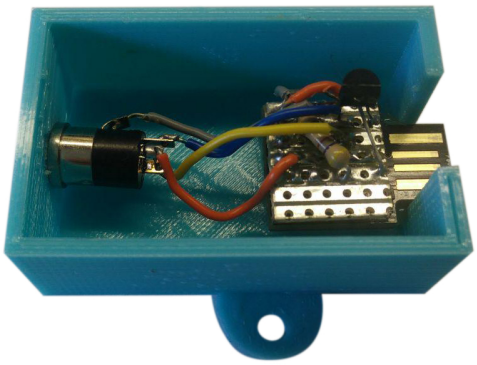

Figura 5.2.: Muntatge final del dispositiu sensor de temperatura dins la capsa 3D.

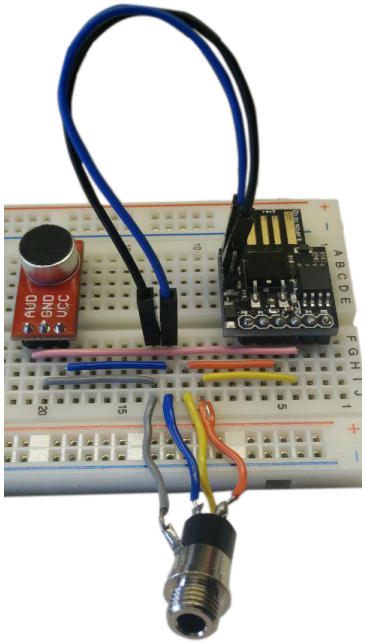

Figura 5.3.: Muntatge final del micròfon electret sobre la placa de prototipatge.

Per comunicar els diferents dispositius, s'ha soldat els connectors mascle a un cable apantallat de 3 contactes més la malla de massa (figura 5.4). S'han fet dos cables, un de curt () i un de més llarg (), per verificar que el protocol TTL Serial funciona bé i era l'elecció correcta.

Figura 5.4.: Contactes del connector mascle amb el cable.

Un cop s'han comprovat els contactes dels diferents dispositius i cables, podem procedir a realitzar les mesures. Connectem els dispositius sensors al gestor i un cop ens apareixen en la web els configurem. Cal comentar, que a l'enviar ordres al gestor a través de la web, sembla que el mòdul WiFi a vegades no respon a la primera i cal clicar més d'una vegada el botó de la web. S'ha observat que aquest comportament s'ha accentuat amb el temps, ja que al principi l'ESP8266 quasi sempre responia. S'ha utilitzat el temps de mostreig de 60 segons per agafar les mostres. S'ha intentat realitzar mesures durant un període de 24 hores, però s'ha observat que al cap de 16 o 18 hores, els sensors han deixat de respondre. Simplement parant i tornant a encendre el gestor, hem vist que es detectaven els sensors de manera normal, per tant ens indica que no és problema de les connexions en les caixes. Usant un multímetre, s'ha detectat que el gestor no estava alimentant els ports dels sensors, però sí que ha respost a l'ordre de parar el mostreig que li hem enviat a través de la web. No s'ha pogut detectar la font del problema, però segurament a causa del fet que el sistema físic del qual es disposa és un prototip, hi ha algun aspecte que no podem controlar al 100%.

En les gràfiques 5.5 i 5.6 es poden observar els valors recollits dels dos sensors. Per poder tenir una referència del significat de les dades del micròfon, s'ha utilitzat un sonòmetre per comparar el nivell de dB amb el valor obtingut. S'ha pogut observar (taula 5.1) que el micròfon electret proporciona valors poc exactes en relació a la lectura del sonòmetre, tot i això, sí que pot ser útil per fer-se una idea del nivell de soroll. Observant la gràfica 5.5 veiem que la temperatura no varia gaire degut a que les mesures s'han fet dins d'una habitació, però sí que es pot veure com la temperatura descendeix a mesura que ens apropem a la nit. Cal dir que la baixada de temperatura observada en el gràfic entre les hores 10 i 11 és a causa que va ploure i es va comprovar presencialment que efectivament va descendir la temperatura. En el cas del gràfic 5.6, veiem que la majoria de valors estan situats entre 0 i 70. Si mirem la taula 5.1, veiem que aquests valors es corresponen entre 45 dB i 60 dB, per tant ens està indicant una situació de poc soroll. Els pics que s'observen poden ser deguts al fet que es produeixi un so elevat just en el moment de la mesura, com per exemple que passi un autobús pel carrer.

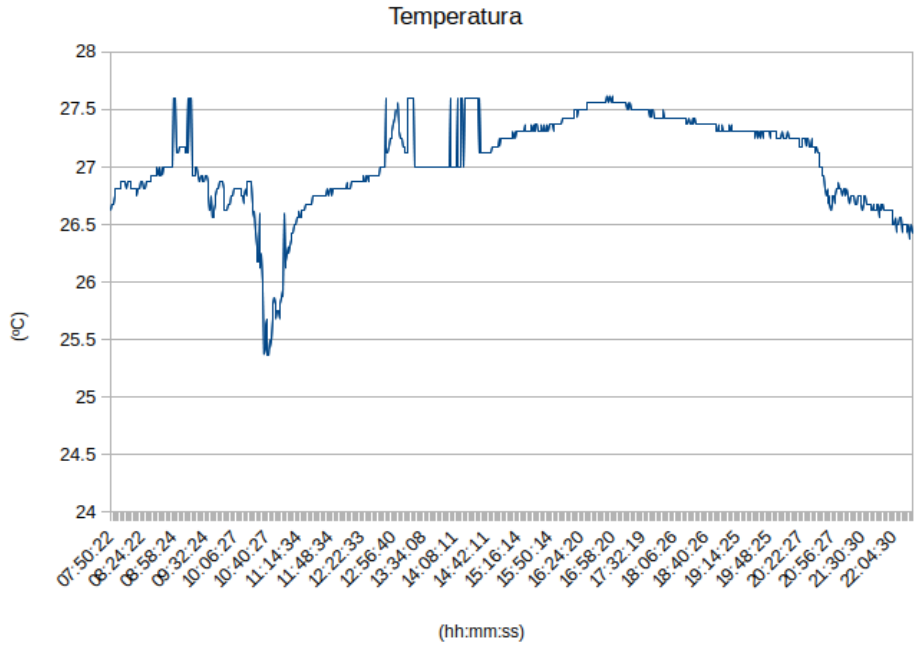

Figura 5.5.: Gràfica de la temperatura recollida pel dispositiu sensor.

Sonòmetre (dB)	Dispositiu sensor
45 - 50	8-30
50-60	30-80
60-70	80-120
70+	120+

Taula 5.1.: Taula comparativa valors sonòmetre i micròfon electret.

Per acabar, s'ha mesurat el consum de les diferents parts del sistema. A la taula 5.2, es pot observar que l'element que consumeix més amb diferència és el mòdul WiFi, però gràcies a l'ús

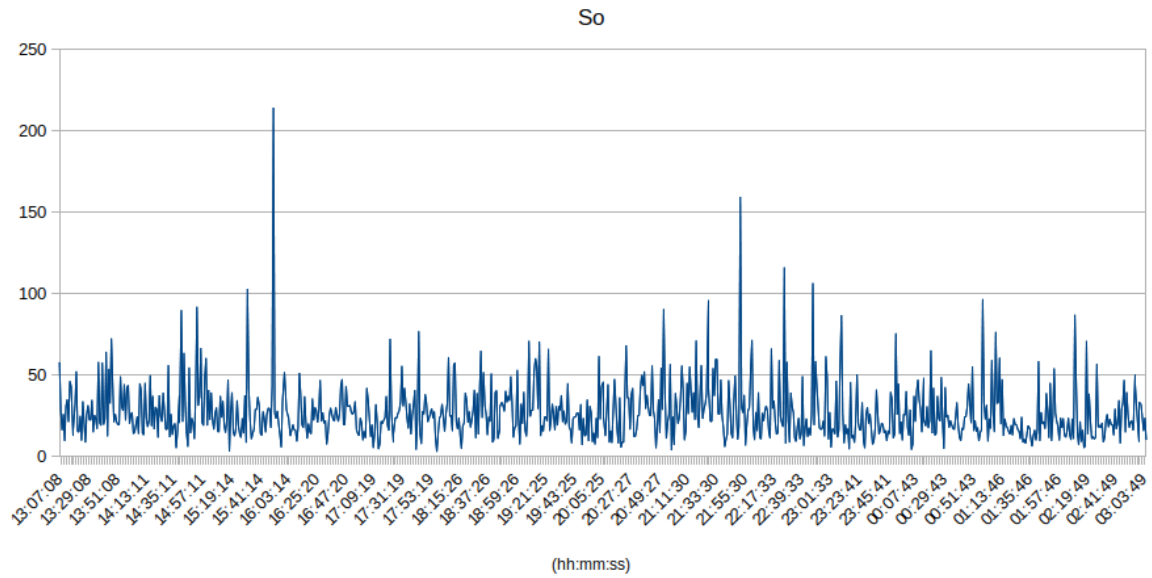

Figura 5.6.: Gràfica del so recollit pel dispositiu sensor.

del mode *deep_sleep* podem reduir el consum dràsticament. En total, el consum d'un dispositiu gestor amb 4 dispositius sensors de so (que estan encesos durant 5 segons), mostrejant cada minut és el següent:

$$30mA + (\frac{5s}{60s} * 22mA) * 4 + \frac{10s}{60s} * 120mA = 57,33mA$$

Amb aquest consum podem tenir el gestor mostrejant durant 7 dies seguits amb una bateria de 10.000 mAh.

Element	Consum
Arduino Uno	`29.8 mA`
Digispark (16MHz)	`35 mA`
Digispark (8MHz)	`31.2 mA`
Digispark (8MHz + optimitzacions)	`29.4 mA`
Digispark (alimentat per Arduino)	`22 mA`
ESP8266 (actiu)	`110 mA - 130 mA`
ESP8266 (deep_sleep)	<200,00 µA
Micròfon Electret	<200,00 µA

Taula 5.2.: Taula de consum dels diferents elements.

6. Conclusions

Amb aquest projecte s'ha aconseguit desenvolupar un sistema de medicions portable i apte per ser utilitzat per persones no expertes. Tot i no haver aconseguit configurar totalment bé els sensors de llum i so, el resultat final ha estat satisfactori.

La realització d'aquest projecte m'ha permès treballar en tots els àmbits de l'Enginyeria en Sistemes TIC. Gràcies a això he pogut aprofundir en aspectes que fins ara no dominava, com per exemple les soldadures de tots els contactes i dispositius, i explorar noves maneres d'utilitzar eines que ja coneixia, com Flask o Docker. També he pogut aprendre coses noves com el procés de disseny i impressió d'objectes 3D.

En definitiva, crec que gràcies a aquest projecte he pogut adquirir noves habilitats i he consolidat els coneixements que m'ha proporcionat l'Enginyeria TIC.

6.1. Propostes de millora

El sistema descrit no deixa de ser un prototip, per tant es poden realitzar diverses millores, tant en hardware com software. Tot i això, les millores que s'exposen a continuació no afecten de manera contundent a la base del sistema, és a dir que el funcionament general continua sent el mateix amb millores o sense:

- Dissenyar una placa de circuit imprès tant pel gestor com per cadascun dels sensors. Això simplificaria molt les connexions dins la capsa 3D i reduiria el consum dels dispositius considerablement, ja que només s'utilitzarien els components imprescindibles.
- Es podrien afegir LEDs en els gestors per indicar l'estat en què està de manera visual sense haver de comprovar-ho en la pàgina web.
- Utilitzar sempre el mateix nom i contrasenya pel punt d'accés és molt insegur. Per solucionar-ho es podria donar l'opció de modificar aquests valors a través de l'aplicació web. En els gestors es podria fer servir la memòria EEPROM (Electrically Erasable Programmable Read-Only Memory) per emmagatzemar les credencials del punt d'accés.
- El micròfon electret porta problemes a l'hora de determinar un valor de so ambient fiable, per tant es podria buscar un altre component més adequat per aquesta tasca.
- Per millorar l'experiència d'usuari es podria fer ús de tecnologies com Javascript o Websockets per no haver de refrescar la pàgina per veure si algun valor ha canviat.

Bibliografia

[Ada17] Adafruit. *Adafruit TSL2591 High Dynamic Range Digital Light Sensor*. Jun. de 2017. URL: https://www.adafruit.com/product/1980.

[AG15] Ams AG. *TSL2591. Light-to-Digital Converter*. Ver. 2.00. Ams AG. Jun. de 2015. URL: http://docs-europe.electrocomponents.com/webdocs/14cd/0900766b814cdbc5.pdf.

[Ard17a] Arduino. *Arduino Uno Rev3*. Jun. de 2017. URL: https://www.arduino.cc/en/main/arduinoBoardUno.

[Ard17b] Arduino. *Serial*. Jun. de 2017. URL: https://www.arduino.cc/en/Reference/Serial.

[Ard17c] Arduino. *SoftwareSerial Library*. Jun. de 2017. URL: https://www.arduino.cc/en/Reference/SoftwareSerial.

[Atm13] Atmel. *ATtiny25/V / ATtiny45/V / ATtiny85/V*. Ver. 2586Q. Atmel. Ago. de 2013. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf.

[Atm15] Atmel. *ATmega48A/PA/88A/PA/168A/PA/328/P*. Atmel. Nov. de 2015. URL: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf.

[Atm17] Atmel. *tinyAVR Microcontrollers*. Jun. de 2017. URL: http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx.

[Bel12] Belden. *8723 Multi-Conductor - Shielded Twisted Pair Cable*. Ver. 3. Belden. Feb. de 2012. URL: http://www.farnell.com/datasheets/1712701.pdf.

[btn17] btn-stuff. *tutum-docker-mongodb*. Jun. de 2017. URL: https://github.com/raft-buildpacks-arm/mongodb.

[Cor17] Corelis. *SPI Interface*. Jun. de 2017. URL: http://www.corelis.com/education/SPI_Tutorial.htm.

[Dig17] Digistump. *Digistump Wiki*. Jun. de 2017. URL: https://digistump.com/wiki/digispark.

[Doc17a] Docker. *Docker Compose*. Jun. de 2017. URL: https://docs.docker.com/compose/.

[Doc17b] Docker. *Docker Documentation*. Jun. de 2017. URL: https://docs.docker.com/.

[Doc17c] Docker. *Docker Hub*. Jun. de 2017. URL: https://hub.docker.com/.

[Ega17] Kelly Egan. *Arduino OpenSCAD mounting library*. Jun. de 2017. URL: https://www.thingiverse.com/thing:64008.

[Ele10] Challenge Electronics. *Omni-Directional Foil Electret Condenser Microphone*. Challenge Electronics. Gen. de 2010. URL: https://cdn.sparkfun.com/datasheets/Sensors/Sound/CEM-C9745JAD462P2.54R.pdf.

[Fou17] Raspberry Pi Foundation. *Raspberry Pi 3 Model B*. Jun. de 2017. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[Gam17] Nick Gammon. *SPI - Serial Peripheral Interface - for Arduino*. Jun. de 2017. URL: https://www.gammon.com.au/spi.

[Ins08] Texas Instruments. *Low Power, Single-supply, Rail-to-rail Operational Amplifiers*. Texas Instruments. Jul. de 2008. URL: http://www.ti.com/lit/ds/symlink/opa344.pdf.

[Ins17] National Instruments. *NI Wireless Sensor Network (WSN) Starter Kit*. National Instruments. Jun. de 2017. URL: http://www.ni.com/pdf/products/us/350909A-01_WSN_guide_lr.pdf.

[Int17a] Maxim Integrated. *Add Control, Memory, Security, and Mixed-Signal Functions with a Single Contact*. Jun. de 2017. URL: https://www.maximintegrated.com/en/app-notes/index.mvp/id/3989.

[Int17b] Maxim Integrated. *DS18B20*. Maxim Integrated. Mar. de 2017. URL: http://www.farnell.com/datasheets/1917635.pdf.

[Jan17] Jan-Willem. *Digispark case and cover*. Jun. de 2017. URL: https://www.thingiverse.com/thing:41291.

[Kaa17] Kaa. *Kaa*. Jun. de 2017. URL: https://www.kaaproject.org/.

[Kel17] Simon Kelley. *Dnsmasq*. Jun. de 2017. URL: http://www.thekelleys.org.uk/dnsmasq/doc.html.

[Kug11] Thomas Kugelstadt. *Extending the SPI bus for long-distance communication*. Texas Instruments. Nov. de 2011. URL: http://www.ti.com/lit/an/slyt441/slyt441.pdf.

[Lew17] James Lewis. *millis() Tutorial: Arduino Multitasking*. Jun. de 2017. URL: https://www.baldengineer.com/millis-tutorial.html.

[LIM17] Associació LIMA. *Associació LIMA*. Jun. de 2017. URL: http://lima.cat.

[Mal13] Jouni Malinen. *hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. Gen. de 2013. URL: https://w1.fi/hostapd/.

[Mar17] Phil Martin. *Using Raspberry Pi 3 as a WiFi access point with hostapd*. Jun. de 2017. URL: https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd.

[Max17] MaxDetect. *Digital relative humidity and temperature sensor RHT03*. MaxDetect Technology Co., Ltd. Jun. de 2017. URL: https://cdn.sparkfun.com/datasheets/Sensors/Weather/RHT03.pdf.

[Mon17] MongoDB. *MongoDB Docs*. Jun. de 2017. URL: https://docs.mongodb.com/.

[Ope17] OpenSCAD. *OpenSCAD*. Jun. de 2017. URL: http://www.openscad.org/.

[Rei17] Kenneth Reitz. *Requests: HTTP for Humans*. Jun. de 2017. URL: http://docs.python-requests.org/en/master.

[Ron17a] Armin Ronacher. *Flask Docs*. Ver. 0.12.x. Jun. de 2017. URL: http://flask.pocoo.org/docs/0.12.

[Ron17b] Armin Ronacher. *Jinja2 Docs*. Ver. 2.9. Jun. de 2017. URL: http://jinja.pocoo.org/docs/2.9/.

[Run14] Jay Runkel. *Time Series Data - Part 1, Schema Design*. MongoDB, jun. de 2014. URL: https://www.mongodb.com/presentations/mongodb-time-series-data-part-1-setting-stage-sensor-management.

[Sem08] Nordic Semiconductor. *nRF24L01+. Single Chip 2.4GHz Transceiver*. Ver. 1.0. Nordic Semiconductor. Mar. de 2008. URL: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf.

[Spa17a] Sparkfun. *Serial Communication*. Jun. de 2017. URL: https://learn.sparkfun.com/tutorials/serial-communication.

[Spa17b] Sparkfun. *SparkFun Electret Microphone Breakout*. Jun. de 2017. URL: https://www.sparkfun.com/products/12758.

[Stu10] ITead Studio. *HC-05. Bluetooth to Serial Port Module*. Ver. 1.0. ITead Studio. Jun. de 2010. URL: http://www.electronicaestudio.com/docs/istd016A.pdf.

[Sys17a] Advantic Sys. *STKIT*. Jun. de 2017. URL: https://www.advanticsys.com/shop/stkit-p-29.html?lang=en.

[Sys17b] Espressif Systems. *ESP8266 AT Instruction Set*. Ver. 2.1. Espressif Systems. Mai. de 2017. URL: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf.

[Tea15] Espressif Systems IOT Team. *ESP8266EX Datasheet*. Ver. 4.3. Espressif. Gen. de 2015. URL: http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf.

[Tec10] Avago Technologies. *APDS-9301. Miniature Ambient Light Photo Sensor with Digital (I2C) Output*. Avago Technologies. Gen. de 2010. URL: http://www.farnell.com/datasheets/1816958.pdf.

[Tim17a] Shay Green Tim Boschke. *Micronucleus*. Jun. de 2017. URL: https://github.com/micronucleus/micronucleus.

[Tim17b] Shay Green Tim Boschke. *Micronuvleus Tag v1.11*. Jun. de 2017. URL: https://github.com/micronucleus/micronucleus/tree/v1.11/upgrade/releases.

[Wik17] Wikipedia. *Copy-on-write*. Jun. de 2017. URL: https://es.wikipedia.org/wiki/Copy-on-write.